TUGAS AKHIR – KI141502

IMPLEMENTASI *FAST MINIMUM SPANNING TREE* UNTUK MELAKUKAN PENGELOMPOKAN DATA PADA PENGENALAN POLA

STEVEN FREDIAN ANDY PUTRA
NRP 5111100060

Dosen Pembimbing I
Dr. Eng. Chastine Fatichah, S.Kom., M.Kom.

Dosen Pembimbing II
Diana Purwitasari, S.Kom, M.Sc.

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2015

TUGAS AKHIR – KI141502

IMPLEMENTASI *FAST MINIMUM SPANNING TREE* UNTUK MELAKUKAN PENGELOMPOKAN DATA PADA PENGENALAN POLA

STEVEN FREDIAN ANDY PUTRA
NRP 5111100060

Dosen Pembimbing I
Dr. Eng. Chastine Fatichah, S.Kom., M.Kom.

Dosen Pembimbing II
Diana Purwitasari, S.Kom, M.Sc.

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2015

UNDERGRADUATE THESES – KI141502

IMPLEMENTATION OF FAST MINIMUM SPANNING TREE FOR CLUSTERING ON PATTERN RECOGNITION

STEVEN FREDIAN ANDY PUTRA
NRP 5111100060

Supervisor I
Dr. Eng. Chastine Fatichah, S.Kom., M.Kom.

Supervisor II
Diana Purwitasari, S.Kom, M.Sc.

DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA 2015

LEMBAR PENGESAHAN

IMPLEMENTASI FAST MINIMUM SPANNING TREE UNTUK MELAKUKAN PENGELOMPOKAN DATA PADA PENGENALAN POLA

TUGAS AKHIR

Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Rumpun Mata Kuliah Komputasi Cerdas dan Visi
Program Studi S-1 Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember

Oleh
STEVEN FREDIAN A.P.
NRP : 5111100060

Disetujui oleh Pembimbing Tugas Akhir:

1. Dr. Eng. Chastine Fatichah, S.Kom., M.Kom. .....
   NIP: 19751220 200112 2 002 (Pembimbing1)

2. Diana Purwitasari, S.Kom., M.Sc. .....
   NIP: 19780410 200312 2 001 (Pembimbing 2)

SURABAYA
JUNI, 2015

IMPLEMENTASI *FAST MINIMUM SPANNING TREE* UNTUK MELAKUKAN PENGELOMPOKAN DATA PADA PENGENALAN POLA

Nama Mahasiswa	**:**	**STEVEN FREDIAN ANDY PUTRA**
NRP	**:**	**5111100060**
Jurusan	**:**	**Teknik Informatika FTIF-ITS**
Dosen Pembimbing 1	**:**	**Dr.Eng. Chastine Fatichah, S.Kom., M.Kom.**
Dosen Pembimbing 2	**:**	**Diana Purwitasari, S.Kom , M.Sc**

Abstrak

Pengelompokan data berdasarkan kemiripan pola untuk data non-convex tidak bisa dilakukan dengan algoritma klastering biasa seperti k-means. Data bersifat convex memiliki kondisi jika diambil sembarang pasangan data maka garis yang menghubungkan data termasuk dalam himpunan tersebut. Minimum Spanning Tree (MST) dapat digunakan untuk menyelesaikan permasalahan pengelompokan data non-convex. Hal ini dikarenakan MST merupakan pohon rentang dari suatu graph yang menghubungkan semua titik dengan bobot paling minimal. Akan tetapi implementasi MST konvensional ke dataset yang besar membutuhkan banyak waktu.

Pada Tugas Akhir ini dibentuk solusi untuk menyelesaikan pengelompokan data dengan metode MST yang dimodifikasi. Solusi tersebut mengombinasikan metode Fast Minimum Spanning Tree (FMST) dan klastering berdasarkan MST. FMST menghasilkan MST-perkiraan dengan waktu yang lebih cepat. FMST menggunakan konsep divide and conquer (data dibagi menjadi beberapa kelompok) lalu diselesaikan MST tiap kelompok, dan semuanya digabungkan. Klastering berdasarkan MST menghapus edge yang memiliki bobot melebihi dari threshold yang ditentukan. Pada setiap

perulangan menambah threshold sampai tidak ada edge yang dapat dihapus, dilakukan uji validitas indeks Ray & Turi untuk mengetahui hasil klastering paling optimal.

Uji coba dilakukan pada data sintesis yang telah disiapkan serta data iris dan wine. Hasil uji coba menunjukkan bahwa metode FMST mampu mencari MST pada dataset dengan waktu lebih cepat. Perbedaan jumlah bobot edge pada FMST dengan MST sebenarnya atau weight error rata-rata yang dihasilkan FMST tidak melebihi 1 persen. Klastering FMST mampu mengidentifikasi jumlah klaster dengan benar pada 13 dari 15 uji coba data sintesis yang tidak bersifat convex.

Kata kunci: Pengenalan Pola, Klastering, Convex Set, Minimum Spanning Tree, Fast Minimum Spanning Tree, Indeks Dunn.

IMPLEMENTATION OF FAST MINIMUM SPANNING TREE FOR CLUSTERING ON PATTERN RECOGNITION

Student's Name	**:**	**STEVEN FREDIAN ANDY PUTRA**
Student's ID	**:**	**5111100060**
Department	**:**	**Teknik Informatika FTIF-ITS**
First Advisor	**:**	**Dr.Eng. Chastine Fatichah, S.Kom., M.Kom.**
Second Advisor	**:**	**Diana Purwitasari, S.Kom , M.Sc**

Abstract

Grouping data based on their pattern similarities for non-convex data cannot be done with typical clustering algorithms such as k-means. Convex data is defined with certain characteristic so that a pair of data items could have a connected line which also consists data items included in the dataset. Minimum Spanning Tree (MST) can be used to solve problem of grouping non-convex data. MST is a tree that connects data items with the most minimal weight value. However conventional MST requires a great deal of running time.

A modified MST is implemented in this work called Fast Minimum Spanning Tree (FMST) which combines MST and a clustering method based on MST. FMST could result an approximation of MST outcome with faster running time. FMST used an approach of divide-conquer that divides data into several groups, resolved the MST of each group, and combined all of those MST groups to get the final MST. The MST-based clustering in FMST removes edges with weight value that exceeds a specified threshold value. Ray & Turi index value is used to determine the most optimal clustering

results in each iteration process for adjusting the threshold value until there are no more edges that can be removed.

The experiments used synthetic data which has been prepared beforehand as well as standard data (Iris and Wine dataset). The experiment indicator is weight error value which means the differences in the summation value of edge weight for FMST result and the real MST result. The experiment in this work showed less than one percent weight error value. FMST could identify cluster numbers correctly in 13 of the 15 trials of the synthetic data.

Keywords: Pattern Recognition, Clustering, Convex Set, Minimum Spanning Tree, Fast Minimum Spanning Tree, Index Dunn

KATA PENGANTAR

Segala puji dan syukur kepada Tuhan Yang Maha Esa yang telah memberikan rahmat dan karunia-Nya sehingga penulis dapat menyelesaikan tugas akhir yang berjudul "*Implementasi Fast Minimum Spanning Tree untuk Melakukan Pengelompokan Data pada Pengenalan Pola*".

Tugas Akhir ini diajukan untuk memenuhi salah satu syarat memperoleh gelar Sarjana Komputer di Jurusan Teknik Informatika Fakultas Teknologi Informasi Institut Teknologi Sepuluh Nopember.

Dengan selesainya Tugas Akhir ini, penulis berharap apa yang penulis telah kerjakan dapat memberikan manfaat bagi perkembangan ilmu pengetahuan serta bagi penulis.

Keberhasilan dalam penyelesaian tugas akhir ini tidak terlepas dari bantuan berbagai pihak. Maka dari itu, pada kesempatan ini penulis ingin menyampaikan rasa terima kasih yang tulus dan sebesar-besarnya kepada semua pihak, terutama kepada:

1. Tuhan Yang Maha Esa yang telah memberikan rahmat dan karunia-Nya sehingga penulis dapat menyelesaikan Tugas Akhir dengan baik.
2. Keluarga penulis (Papa Andy, Mama Ririn, dan adik Pipin) yang senantiasa memberikan doa dan dukungan baik secara moral maupun finansial selama penulis menyelesaikan Tugas Akhir.
3. Bapak Dr. Agus Zainal Arifin, S.Kom, M.Kom selaku dosen wali penulis, yang selalu memberikan bimbingan, dukungan, serta motivasi selama masa perkuliahan di Teknik Informatika ITS.
4. Ibu Dr.Eng. Chastine Fatichah, S.Kom., M.Kom., selaku dosen pembimbing I atas bimbingan, ilmu, kesabaran dan bantuan-bantuan berharga lainnya sehingga penulis dapat menyelesaikan tugas akhir ini tepat pada waktunya.

5. Ibu Diana Purwitasari, S.Kom, M.Sc. selaku dosen pembimbing II atas bimbingan, bantuan dan kesabaran selama proses pengerjaan tugas akhir.
6. Bapak dan Ibu dosen dan karyawan jurusan Teknik Informatika ITS yang telah banyak mengajarkan ilmu pada penulis selama menempuh studi di Teknik Informatika ITS.
7. Seluruh teman penulis, khususnya teman-teman Angkatan 2011 dalam menempuh perkuliahan di Jurusan Teknik Informatika Fakultas Teknologi Informasi Institut Teknologi Sepuluh Nopember.
8. Dan seluruh pihak yang turut membantu dalam pengerjaan tugas akhir ini, baik secara moril maupun materil, yang tidak dapat penulis sebutkan satu per satu di sini.

Penulis berharap bahwa tugas akhir ini dapat memberikan manfaat pada semua pihak khususnya penulis sendiri dan *civitas academica* Teknik Informatika ITS. Penulis mohon maaf bila terdapat kesalahan, kelalaian maupun kekurangan dalam penyusunan tugas akhir ini. Kritik dan saran yang membangun dapat disampaikan sebagai bahan perbaikan kedepan.

Surabaya, Juni 2015
Penulis

DAFTAR ISI

[Halaman ini sengaja dikosongkan]

DAFTAR TABEL

[Halaman ini sengaja dikosongkan]

DAFTAR GAMBAR

DAFTAR KODE SUMBER

[Halaman ini sengaja dikosongkan]

BIODATA PENULIS

Steven Fredian Andy Putra, lahir di Pasuruan, pada tanggal 1 April 1994. Penulis menempuh pendidikan mulai dari SDK Sang Timur Pasuruan (1999-2005), SMPK Sang Timur Pasuruan (2005-2008), SMAN 1 Pasuruan (2008-2011) dan S1 Teknik Informatika ITS (2011-2015).

Selama masa kuliah, penulis aktif dalam organisasi Himpunan Mahasiswa Teknik Computer (HMTC). Diantaranya adalah menjadi staff departemen Dalam Negeri Himpunan Mahasiswa Teknik Computer ITS 2012-2013 dan staff ahli Dalam Negeri Himpunan Mahasiswa Teknik Computer ITS 2013-2014. Penulis juga aktif dalam kegiatan kepanitiaan Schematics. Diantaranya penulis pernah menjadi sttaff seminar nasional NST Schematics 2012 dan staff Humas Schematics 2013.

Selama kuliah di teknik informatika ITS, penulis mengambil bidang minat Komputasi Cerdas Visual (KCV). Komunikasi dengan penulis dapat melalui email: **steven.fredian@gmail.com**.

BAB I
PENDAHULUAN

1.1 Latar Belakang

Pengenalan pola adalah suatu kemampuan untuk mengelompokkan obyek berdasarkan parameter-parameter yang telah ditentukan kedalam sejumlah kategori atau kelas. Pengenalan pola dapat dibagi menjadi dua, yaitu *supervised*, dan *unsupervised*. *Supervised* adalah pembelajaran yang memiliki suatu contoh, sehingga hasil dari pembelajaran ini mengacu pada contoh yang diberikan. Sebaliknya, *unsupervised* adalah pembelajaran yang tidak memiliki contoh, sehingga hasil dari pembelajaran ini menggunakan prosedur yang berusaha untuk mencari partisi dari sebuah pola [1]. Salah satu implementasi *unsupervised learning* adalah klastering.

Klastering merupakan suatu proses pembagian dari suatu kumpulan data menjadi kelompok – kelompok yang lebih kecil berdasarkan kesamaan fitur [2]. Klastering memegang peranan penting di beberapa bidang, seperti *image processing*, komputasi biomedik, ekonomi, dan lain- lain. Sebagai salah satu contoh di bidang *image processing*, klastering dapat digunakan untuk segmentasi citra berdasarkan warna [3]. Citra dapat dibagi ke beberapa kelompok berdasarkan kemiripan warna di dalamnya.

Kemajuan teknologi saat ini menyebabkan berkembangnya data – data secara pesat. Menurut macamnya, data dibagi menjadi dua, yaitu data real, dan data sintesis. Data sintesis merupakan data buatan yang didesain mengikuti suatu kondisi tertentu yang tidak dapat ditemukan di data real [4]. Dalam hal ini, data sintesis dibentuk menyerupai pola seperti spiral, curve, dan lain-lain. Data tersebut disebut *Convex Set*. Algoritma klastering yang menggunakan konsep jarak terdekat dengan *centroid* seperti K-Means tidak dapat menyelesaikan masalah pengelompokan data yang bersifat *non-Convex* [5].

Untuk mengatasi masalah tersebut, maka akan digunakan klastering dengan menggunakan *Minimum Spanning Tree* (*MST*). Akan tetapi, sulit untuk mengimplementasikan algoritma konvensional untuk menyelesaikan permasalahan *MST* ke *dataset* yang besar, dikarenakan kompleksitas yang tinggi [6]. Maka dari itu, akan digunakan *Fast Minimum Spanning Tree* (*FMST*) untuk mendapatkan *MST* dengan waktu lebih singkat. *FMST* menggunakan teknik *Divide and Conquer*, dimana *dataset* dibagi ke beberapa *subset* menggunakan K-Means, lalu diselesaikan permasalahan *MST* nya tiap *subset* terlebih dahulu. Kemudian, *MST* tiap *subset* dihubungkan untuk membentuk suatu *MST*. *Set*elah didapatkan hasil *FMST* dari suatu data, klastering dilakukan dengan cara menghapus *edge* dari *FMST* yang bernilai lebih besar dari *threshold*.

Hasil klastering menggunakan *FMST* ini diperoleh dengan waktu yang lebih singkat dan mencapai akurasi yang baik.

1.2 Rumusan Masalah

Rumusan masalah yang diangkat dalam Tugas Akhir ini dapat dipaparkan sebagai berikut:

1. Bagaimana melakukan pengelompokan data menggunakan *FMST*?
2. Bagaimana mengevaluasi kinerja dari pengelompokan data menggunakan *FMST*?

1.3 Batasan Masalah

Permasalahan yang dibahas dalam Tugas Akhir ini memiliki beberapa batasan, yaitu sebagai berikut:

1. Data yang digunakan merupakan data sintesis (2 dimensi/fitur) dan *dataset* lain (*iris, wine)* bersifat numerik.
2. Algoritma *MST* yang digunakan adalah Algoritma Prim.
3. Algoritma *FMST* yang digunakan adalah metode yang diusulkan Caiming Zhong, dkk [6].

4. Algoritma Klastering berdasarkan *MST* yang digunakan adalah algoritma yang diusulkan oleh Prasanta Jana dan Azad Naik [7].

1.4 Tujuan

Tugas akhir ini adalah mengimplementasikan *Fast Minimum Spanning Tree* untuk melakukan pengelompokan data.

1.5 Manfaat

Manfaat yang dapat diambil dari pengerjaan tugas akhir ini yaitu dapat menyelesaikan pengelompokan data yang bersifat *non-Convex* dengan baik.

1.6 Metodologi

Tahapan-tahapan yang dilakukan dalam pengerjaan Tugas Akhir ini adalah sebagai berikut:

1. Penyusunan proposal Tugas Akhir.
   Tahap awal untuk memulai pengerjaan Tugas Akhir adalah penyusunan proposal Tugas Akhir. Proposal Tugas Akhir yang diajukan memiliki gagasan yang sama dengan Tugas Akhir ini, yaitu implementasi metode *FMST, dan* klastering berdasarkan *MST*.

2. Studi literatur
   Pada tahap ini dilakukan pencarian, pengumpulan, pembelajaran dan pemahaman informasi dan literatur yang diperlukan untuk pembuatan implementasi metode *FMST* dan klastering berdasarkan *MST*. Informasi dan literatur didapatkan dari literatur buku dan sumber-sumber informasi lain yang berhubungan.

3. Analisis dan desain perangkat lunak

   Tahap ini meliputi perancangan sistem berdasarkan studi literatur dan pembelajaran konsep teknologi dari perangkat lunak yang ada. Tahap ini mendefinisikan alur dari implementasi. Langkah-langkah yang dikerjakan juga didefinisikan pada tahap ini. Pada tahapan ini dibuat *prototype* sistem, yang merupakan rancangan dasar dari sistem yang akan dibuat. Serta dilakukan desain suatu sistem dan desain proses-proses yang ada.

4. Implementasi perangkat lunak

   Implementasi merupakan tahap membangun rancangan program yang telah dibuat. Pada tahapan ini merealisasikan apa yang terdapat pada tahapan sebelumnya, sehingga menjadi sebuah program yang sesuai dengan apa yang telah direncanakan.

5. Pengujian dan evaluasi

   Pada tahapan ini dilakukan uji coba pada data yang telah dikumpulkan. Pengujian dan evaluasi akan dilakukan dengan menggunakan MATLAB. Tahapan ini dimaksudkan untuk mengevaluasi kesesuaian data dan program serta mencari masalah yang mungkin timbul dan mengadakan perbaikan jika terdapat kesalahan.

6. Penyusunan buku Tugas Akhir.

   Pada tahapan ini disusun buku yang memuat dokumentasi mengenai pembuatan serta hasil dari implementasi perangkat lunak yang telah dibuat.

1.7 Sistematika Penulisan Laporan Tugas Akhir

Buku Tugas Akhir ini bertujuan untuk mendapatkan gambaran dari pengerjaan Tugas Akhir ini. Selain itu, diharapkan dapat berguna untuk pembaca yang tertarik untuk melakukan

pengembangan lebih lanjut. Secara garis besar, buku Tugas Akhir terdiri atas beberapa bagian seperti berikut ini:

Bab I Pendahuluan
Bab yang berisi mengenai latar belakang, tujuan, dan manfaat dari pembuatan Tugas Akhir. Selain itu permasalahan, batasan masalah, metodologi yang digunakan, dan sistematika penulisan juga merupakan bagian dari bab ini.

Bab II Dasar Teori
Bab ini berisi penjelasan secara detail mengenai dasar-dasar penunjang dan teori-teori yang digunakan untuk mendukung pembuatan Tugas Akhir ini.

Bab III Perancangan Perangkat Lunak
Bab ini berisi tentang desain sistem yang disajikan dalam bentuk *pseudocode*.

Bab IV Implementasi
Bab ini membahas implementasi dari desain yang telah dibuat pada bab sebelumnya. Penjelasan berupa *code* yang digunakan untuk proses implementasi.

Bab V Uji Coba Dan Evaluasi
Bab ini menjelaskan kemampuan perangkat lunak dengan melakukan pengujian kebenaran dan pengujian kinerja dari sistem yang telah dibuat.

Bab VI Kesimpulan Dan Saran
Bab ini merupakan bab terakhir yang menyampaikan kesimpulan dari analisa hasil uji coba yang dilakukan

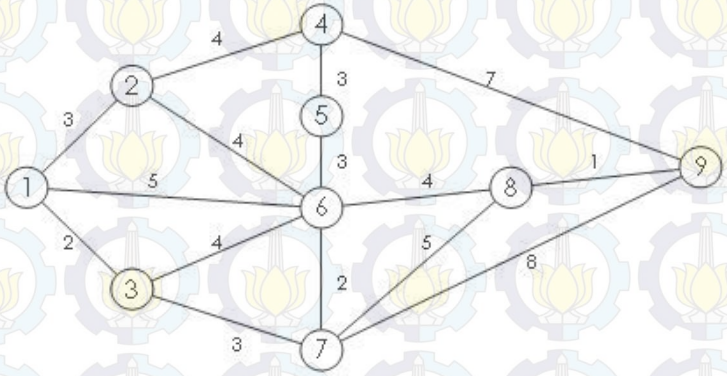
dan saran untuk pengembangan perangkat lunak ke depannya.

BAB II
TINJAUAN PUSTAKA

Bab ini berisi penjelasan teori-teori yang berkaitan dengan algoritma yang diajukan pada pengimplementasian program. Penjelasan ini bertujuan untuk memberikan gambaran secara umum terhadap program yang dibuat dan berguna sebagai penunjang dalam pengembangan perangkat lunak.

2.1 *Graph*

Graph G = (V, E) merupakan himpunan yang terdiri dari *vertex V* dan *edge E*. *Edge (u, v)* merupakan *edge* yang menghubungkan *vertex u* dan *vertex* v [8]. Jika *edge* pada *graph* memiliki bobot (berupa bilangan real, menandakan *cost*, *length*), maka disebut *weighted graph* [8]. *Graph* yang terdiri dari *edge* yang menghubungkan seluruh kombinasi pasangan data disebut *complete graph* [9]. Ilustrasi *graph* ditunjukkan pada Gambar 2.1 [10].

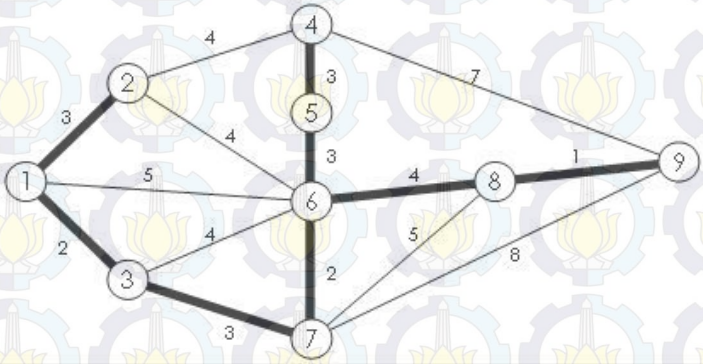


Gambar 2.1. Ilustrasi Graph

2.2 *Minimum Spanning Tree*

Minimum Spanning Tree (MST) adalah suatu *graph* yang memiliki batasan dimana semua *vertex* dalam *graph* terhubung tanpa terdapat *cycle* didalamnya dan memiliki total bobot *edge* yang paling minimum. *Cycle* merupakan rute dalam *graph* yang berawal dan berakhir pada *vertex* yang sama [11]. *Minimum Spanning Tree* banyak digunakan ketika ditemui masalah dalam hal penentuan seperti panjang kabel, khususnya dalam dunia kelistrikan, Jaringan (LAN), PDAM (penentuan panjangnya pipa) [12].

Sebagai contoh, Gambar 2.1 adalah denah saluran listrik pada suatu perusahaan. Teknisi listrik akan menyalurkan listrik dari ruang bagian depan sampai keseluruh ruangan dengan total panjang kabel yang seefisien mungkin. Ilustrasi *MST* dari Gambar 2.1 ditunjukkan pada Gambar 2.2 [10].

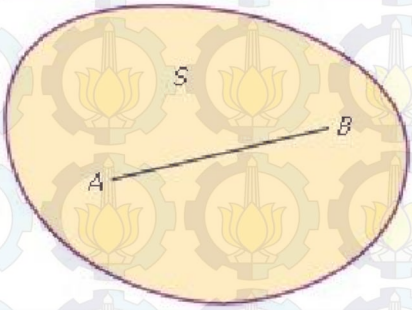


Gambar 2.2. Minimum Spanning Tree dari Gambar 2.1.

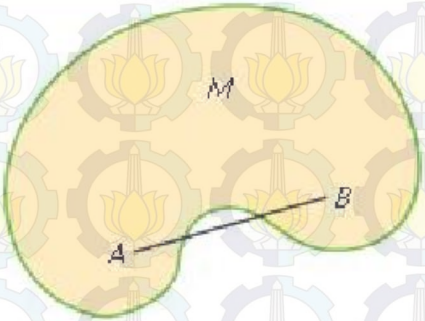
2.3 *Convex Set*

Convex Set adalah *set* atau himpunan titik, dimana setiap titik didalamnya dihubungkan oleh garis yang termasuk didalam *set* tersebut [13]. Pada Gambar 2.3 [14], ditunjukkan bahwa S

adalah contoh *Convex Set*. Titik A dan B pada S dihubungkan oleh garis yang termasuk dalam S. Sedangkan M pada Gambar 2.4 [14], merupakan contoh *Non-Convex Set*. A dan B pada M dihubungkan oleh garis yang tidak termasuk dalam M.



Gambar 2.3. Ilustrasi Convex Set



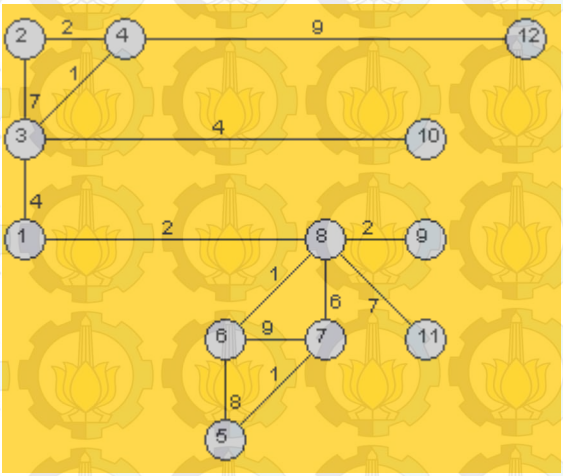
Gambar 2.4. Ilustrasi Non-Convex Set

2.4 Algoritma Prim

Algoritma Prim merupakan salah satu algoritma *greedy* yang dapat menyelesaikan permasalahan *MST* [15]. Diberikan *graph* G = (*U, V)*. Pada awalnya untuk setiap *vertex* $v \in V$

dilakukan inisialisasi *visited(v) = false*. Pilih salah satu *vertex s* $\in$ *V* secara bebas sebagai titik mulai. Set *visited(s) = true*. Kemudian enqueue setiap *edge(s, v)* $\in$ *V* beserta bobotnya pada *minimum weight priority queue D*. Selama *D* tidak kosong, cek *edge(u, s)* yang merupakan *top* dari *D*. *Dequeue top* dari *D*. Jika *visited(s) = false*, ubah *visited(s) = true*. Masukkan *edge(u,s)* kedalam himpunan *mst*, dan *enqueue* setiap *edge(s,v)* $\in$ *V* dengan syarat *visited(s)= false* pada *D*.

Pada akhir operasi, didapatkan himpunan berisi *list edge* yang merupakan *MST* dari *graph G*. Sebagai contoh, pada Gambar 2.5 divisualisasikan suatu *graph* yang ingin dicari *MST* nya dengan menggunakan Algoritma Prim.

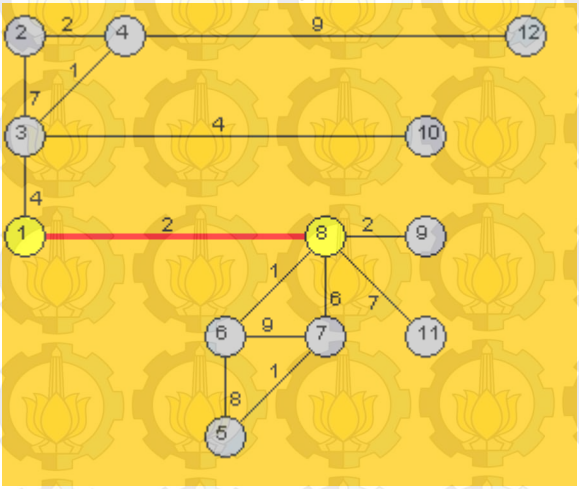


Gambar 2.5. Proses Algoritma Prim

Ditentukan *vertex* 1 sebagai *vertex* awal. Maka set *visited(*1*)= true*. Masukkan semua *edge* yang berawal dari *vertex 1* kedalam *minimum weight priority queue*. *Priority queue* saat ini berisi {1-8:2, 1-3:4}.

Pada *priority queue, top* saat ini adalah *edge(1,8)* dengan bobot 2. Setelah dicek *visited(*8*) = false*, maka set *visited(*8*) = true*, dan masukkan *edge(*1,8*)* ke dalam himpunan *mst*. *Dequeue top* dari

priority queue. Visualisasi sampai tahap ini dapat dilihat pada Gambar 2.6. Masukkan semua *edge* yang berawal dari *vertex* 8 ke *vertex* yang belum pernah dikunjungi. *Priority queue* saat ini berisi {8-6:1, 8-9:2, 1-3:4, 8-7:6, 8-11:7}.

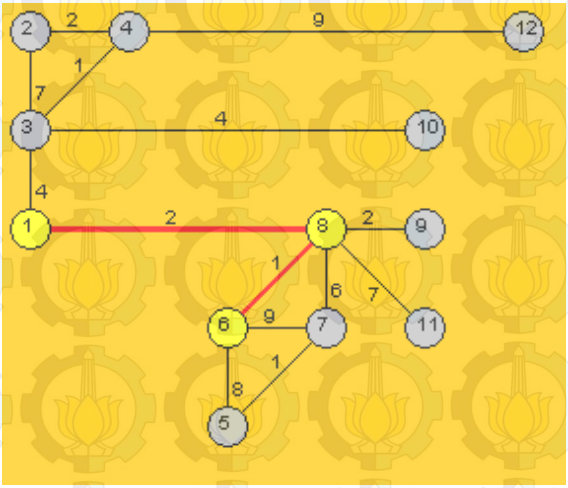


Gambar 2.6. Proses Algoritma Prim (2)

Pada *priority queue, top* saat ini adalah *edge(8,6)* dengan bobot 1. Setelah dicek *visited*(6) = *false*, maka set *visited*(6) = *true*, dan masukkan *edge(*8,6*)* ke dalam himpunan *mst*. *Dequeue top* dari *priority queue*. Visualisasi sampai tahap ini dapat dilihat pada Gambar 2.7. Masukkan semua *edge* yang berawal dari *vertex* 6 ke *vertex* yang belum pernah dikunjungi. *Priority queue* saat ini berisi {8-9:2, 1-3:4, 8-7:6, 8-11:7, 6-5:8, 6-7:9}.

Pada *priority queue, top* saat ini adalah *edge(8,9)* dengan bobot 2. Setelah dicek *visited(*9*)* = *false*, maka set *visited(*9*)* = *true*, dan masukkan *edge(*8,9*)* ke dalam himpunan *mst*. *Dequeue top* dari *priority queue*. Visualisasi sampai tahap ini dapat dilihat pada Gambar 2.8. Masukkan semua *edge* yang berawal dari *vertex* 9 ke *vertex* yang belum pernah dikunjungi. *Priority queue* saat ini berisi {1-3:4, 8-7:6, 8-11:7, 6-5:8, 6-7:9}.

Pada *priority queue*, *top* saat ini adalah *edge(1,3)* dengan bobot 4. Setelah dicek *visited(*4*) = false*, maka set *visited(*4*) = true*, dan masukkan *edge(*1,3*)* ke dalam himpunan *mst*. *Dequeue top* dari *priority queue*. Visualisasi sampai tahap ini dapat dilihat pada Gambar 2.9. Masukkan semua *edge* yang berawal dari *vertex* 3 ke *vertex* yang belum pernah dikunjungi. *Priority queue* saat ini berisi {3-4:1, 3-10:4, 8-7:6, 3-2:7, 8-11:7, 6-5:8, 6-7:9}.

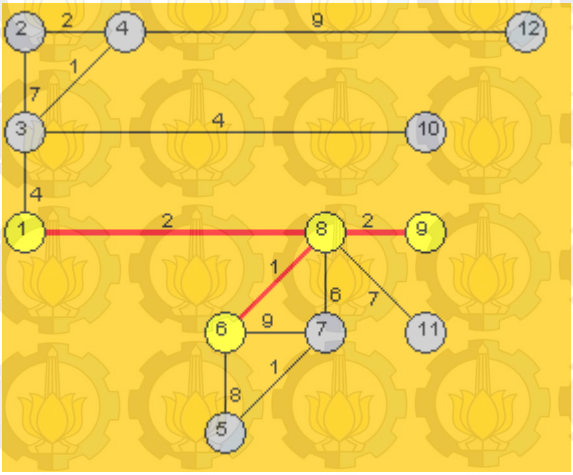


Gambar 2.7. Proses Algoritma Prim (3)

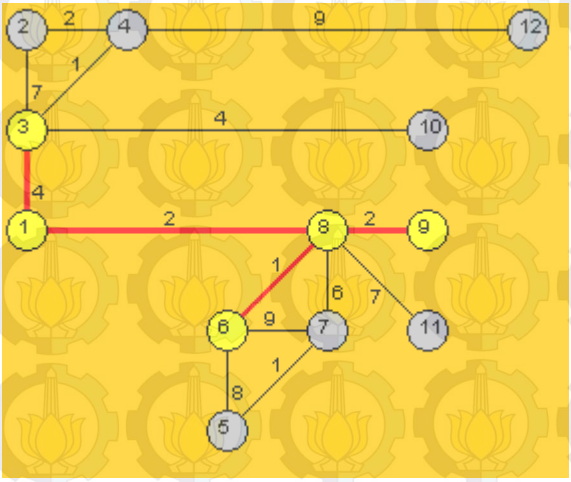
Pada *priority queue*, *top* saat ini adalah *edge(3,4)* dengan bobot 1. Setelah dicek *visited(*4*) = false*, maka set *visited(*4*) = true*, dan masukkan *edge(*3,4*)* ke dalam himpunan *mst*. *Dequeue top* dari *priority queue*. Visualisasi sampai tahap ini dapat dilihat pada Gambar 2.10. Masukkan semua *edge* yang berawal dari *vertex* 4 ke *vertex* yang belum pernah dikunjungi. *Priority queue* saat ini berisi {4-2:2, 3-10:4, 8-7:6, 3-2:7, 8-11:7, 6-5:8, 4-12:9, 6-7:9}.

Pada *priority queue*, *top* saat ini adalah *edge(4,2)* dengan bobot 2. Setelah dicek *visited(*2*) = false*, maka set *visited(*2*) = true*, dan masukkan *edge(*4,2*)* ke dalam himpunan *mst*. *Dequeue top* dari *priority queue*. Visualisasi sampai tahap ini dapat dilihat pada

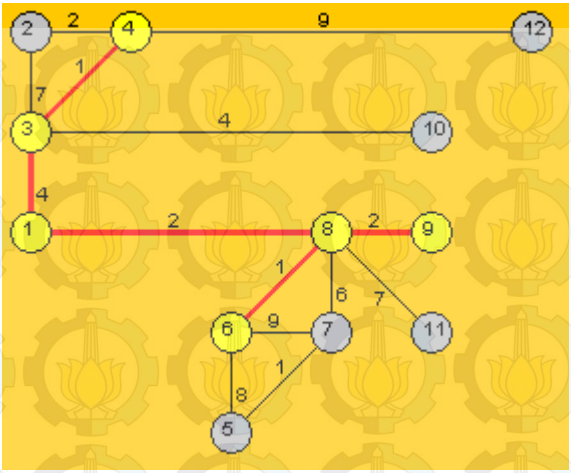
Gambar 2.11. Masukkan semua *edge* yang berawal dari *vertex* 2 ke *vertex* yang belum pernah dikunjungi. *Priority queue* saat ini berisi {3-10:4, 8-7:6, 3-2:7, 8-11:7, 6-5:8, 4-12:9, 6-7:9}.



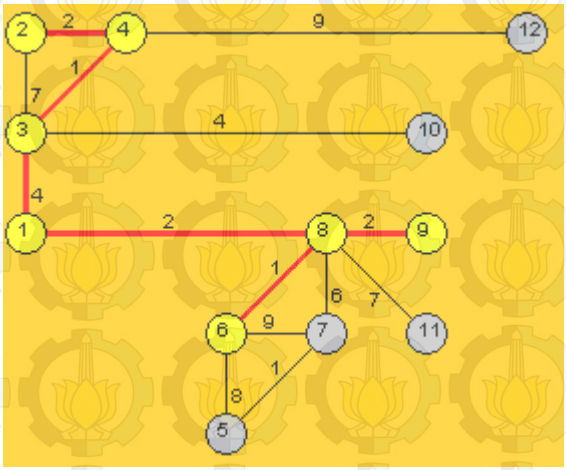
Gambar 2.8. Proses Algoritma Prim (4)



Gambar 2.9. Proses Algoritma Prim (5)



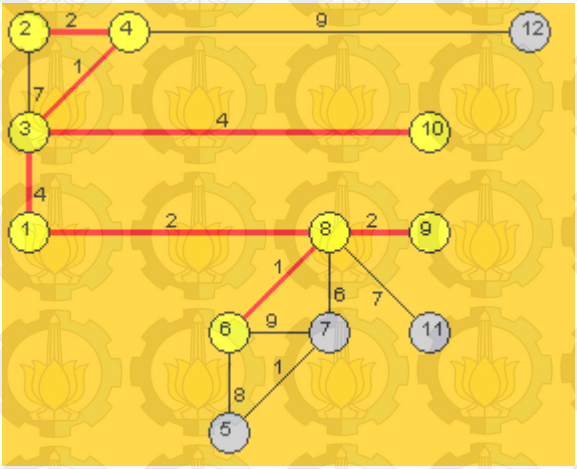
Gambar 2.10. Proses Algoritma Prim (6)



Gambar 2.11. Proses Algoritma Prim (7)

Pada *priority queue*, *top* saat ini adalah *edge(3,10)* dengan bobot 4. Setelah dicek *visited(*10*)* = *false*, maka set *visited(*10*)* = *true*, dan masukkan *edge(*3,10*)* ke dalam himpunan *mst. Dequeue*

top dari *priority queue*. Visualisasi sampai tahap ini dapat dilihat pada Gambar 2.12. Masukkan semua *edge* yang berawal dari *vertex* 10 ke *vertex* yang belum pernah dikunjungi. *Priority queue* saat ini berisi {8-7:6, 3-2:7, 8-11:7, 6-5:8, 4-12:9, 6-7:9}.



Gambar 2.12. Proses Algoritma Prim (8)

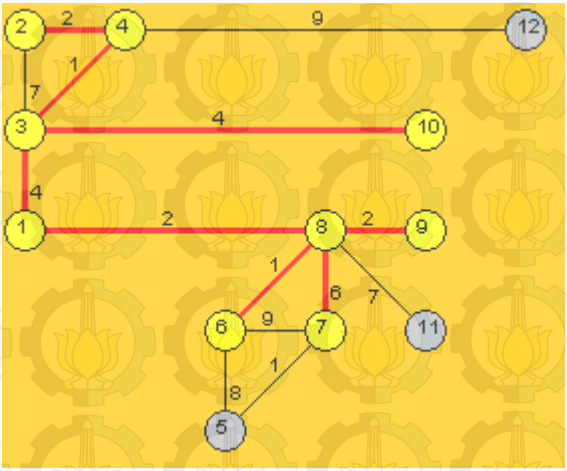
Pada *priority queue, top* saat ini adalah *edge(8,7)* dengan bobot 6. Setelah dicek *visited(7) = false*, maka set *visited(7) = true*, dan masukkan *edge(*8,7*)* ke dalam himpunan *mst*. *Dequeue top* dari *priority queue*. Visualisasi sampai tahap ini dapat dilihat pada Gambar 2.13. Masukkan semua *edge* yang berawal dari *vertex* 7 ke *vertex* yang belum pernah dikunjungi. *Priority queue* saat ini berisi {7-5:1, 3-2:7, 8-11:7, 6-5:8, 4-12:9, 6-7:9}.

Pada *priority queue, top* saat ini adalah *edge(7,5)* dengan bobot 1. Setelah dicek *visited(*5*) = false*, maka set *visited(*5*) = true*, dan masukkan *edge(*7,5*)* ke dalam himpunan *mst*. *Dequeue top* dari *priority queue*. Visualisasi sampai tahap ini dapat dilihat pada Gambar 2.14. Masukkan semua *edge* yang berawal dari *vertex* 5 ke *vertex* yang belum pernah dikunjungi. *Priority queue* saat ini berisi {3-2:7, 8-11:7, 6-5:8, 4-12:9, 6-7:9}.

Pada *priority queue, top* saat ini adalah *edge(3,2)* dengan bobot 7. Setelah dicek *visited(*2*) = true*. *Dequeue top* dari *priority queue*. *Priority queue* saat ini berisi {8-11:7, 6-5:8, 4-12:9, 6-7:9}.

Pada *priority queue, top* saat ini adalah *edge(8,11)* dengan bobot 7. Setelah dicek *visited(*11*) = false*, maka set *visited(*11*) = true*, dan masukkan *edge(*8,11*)* ke dalam himpunan *mst*. *Dequeue top* dari *priority queue*. Visualisasi sampai tahap ini dapat dilihat pada Gambar 2.15. Masukkan semua *edge* yang berawal dari *vertex* 11 ke *vertex* yang belum pernah dikunjungi. *Priority queue* saat ini berisi {6-5:8, 4-12:9, 6-7:9}.

Pada *priority queue, top* saat ini adalah *edge(6,5)* dengan bobot 8. Setelah dicek *visited(*5*) = true*. *Dequeue top* dari *priority queue*. *Priority queue* saat ini berisi {4-12:9, 6-7:9}.

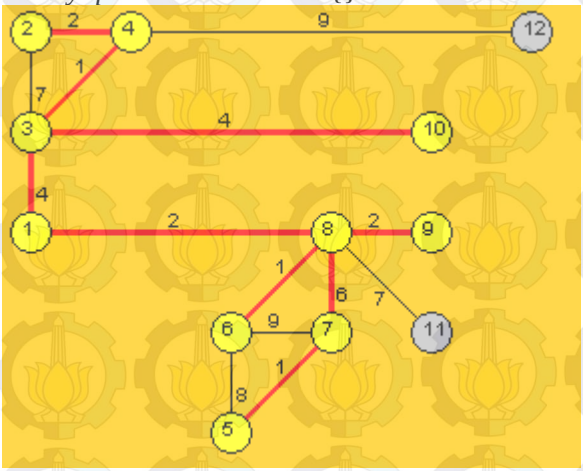


Gambar 2.13. Proses Algoritma Prim (9)

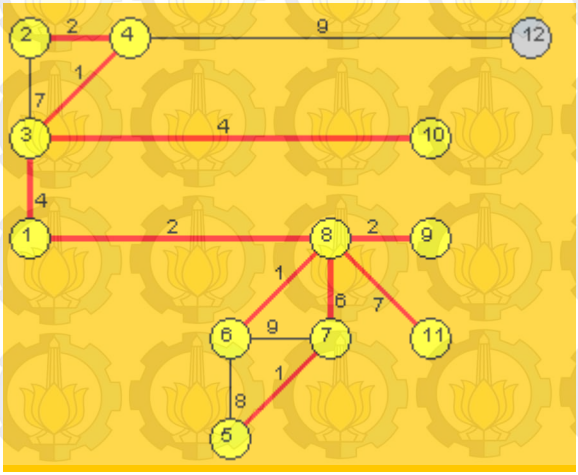
Pada *priority queue, top* saat ini adalah *edge(4,12)* dengan bobot 9. Setelah dicek *visited(*12*) = false*, maka set *visited(*12*) = true*, dan masukkan *edge(*6,5*)* ke dalam himpunan *mst*. *Dequeue top* dari *priority queue*. Visualisasi sampai tahap ini dapat dilihat pada Gambar 2.16. Masukkan semua *edge* yang berawal dari *vertex*

12 ke *vertex* yang belum pernah dikunjungi. *Priority queue* saat ini berisi {6-7:9}.

Pada *priority queue*, *top* saat ini adalah *edge(6,7)* dengan bobot 7. Setelah dicek *visited(*9*) = true*. *Dequeue top* dari *priority queue*. *Priority queue* saat ini berisi {}.

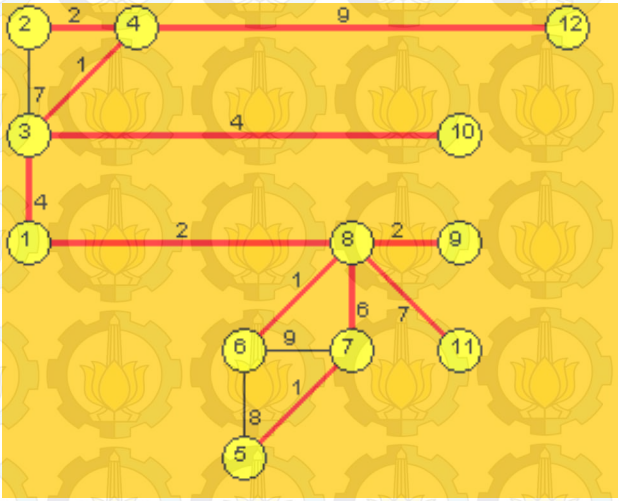


Gambar 2.14. Proses Algoritma Prim (10)



Gambar 2.15. Proses Algoritma Prim (11)

Jika *priority queue* kosong, maka *MST* telah ditemukan. Pada himpunan *mst*, *MST* terdiri dari *edge –edge : 1-8, 8-6, 8-9, 1-3, 3-4, 4-2, 8-7, 3-10, 7-5, 8-11, 4-12*. *Weight*, yang merupakan jumlah dari bobot *edge-edge* pada *MST* tersebut adalah 42.



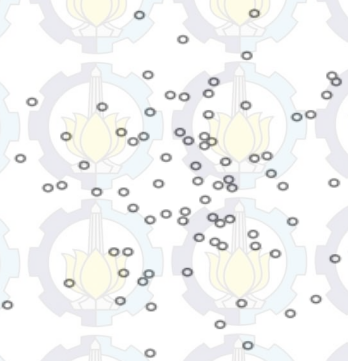
Gambar 2.16. Proses Algoritma Prim (12)

2.5 *Fast Minimum Spanning Tree*

Fast Minimum Spanning Tree (FMST) adalah metode untuk menyelesaikan permasalahan *MST* yang diusulkan oleh Caiming Zhong, dkk [6]. Metode ini tetap menggunakan algoritma konvensional, seperti algoritma Prim. Hanya saja, metode ini menerapkan konsep *Divide and Conquer*, dimana *graph* dibagi menjadi beberapa *subgraph*, dibentuk *MST* - nya masing-masing lalu dirangkai menjadi satu. Hasil yang diberikan oleh metode ini bukan merupakan hasil yang *unique*, tapi merupakan hasil perkiraan atau hasil yang mendekati. Akan tetapi, waktu yang diperlukan dalam metode ini untuk menyelesaikan permasalahan *MST* lebih sedikit dibanding algoritma Prim konvensional.

2.5.1 *Divide and Conquer*

Pada tahap ini, *dataset* dibagi menjadi *k subset*, dimana *k* = √*N*. *Dataset* dibagi menggunakan K-Means yang menggunakan konsep kedekatan *locality*. Setelah data terbagi menjadi *k subset*, hitung jarak tiap data dalam *subset* dengan kombinasi pasangan data. Hal ini menyebabkan tiap *subset* seperti *complete graph* kecil, yang terdiri dari data didalamnya sebagai anggota dari himpunan *vertex*, dan *edge* yang menghubungkan tiap *vertex*. Lalu, algoritma *MST* konvensional diterapkan kepada tiap *subset*, sehingga akan terbentuk *k MST*. Sebagai contoh, diilustrasikan data berukuran 2 dimensi pada Gambar 2.17.

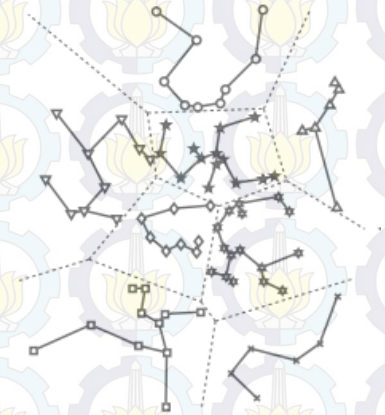


Gambar 2.17. Ilustrasi Dataset

Dataset pada Gambar 2.17 memiliki 78 data dibagi menjadi √78 kelompok, yaitu 8 kelompok menggunakan K-Means. Ilustrasi tersebut ditunjukkan pada Gambar 2.18. Pada Gambar 2.18, ada bulatan hitam yang merupakan *centroid* / titik pusat dari *subset*.

8 *subset* tersebut dicari *MST* nya dengan menggunakan algoritma konvensional *MST*. Sehingga terbentuk 8 *MST subset*. Ilustrasi tersebut dapat dilihat pada Gambar 2.19.

Gambar 2.18. Partisi Dataset ke beberapa subset



Gambar 2.19. Tiap subset dibentuk MST nya

2.5.2 *Combine Subset Algorithm*

Pada tahap ini, *k MST* yang terbentuk akan digabungkan. *Subset* yang digabungkan adalah *subset* yang bertetangga. *Subset* yang bertetangga dihubungkan dengan sebuah *edge* penghubung

yang ditentukan dengan cara di tahap selanjutnya, *Detecting the Connecting Edge*. *Subset* yang bertetangga merupakan *subset* yang *centroid*-nya dihubungkan dengan *MST centroid*.

Sebagai contoh, ditunjukkan pada Gambar 2.20, *dataset* terpartisi ke beberapa *subset*. Ditunjukkan bahwa setiap *subset* dicari *centroid*-nya dengan perwakilan bulatan hitam. *Centroid* merupakan titik tengah dari *subset.*

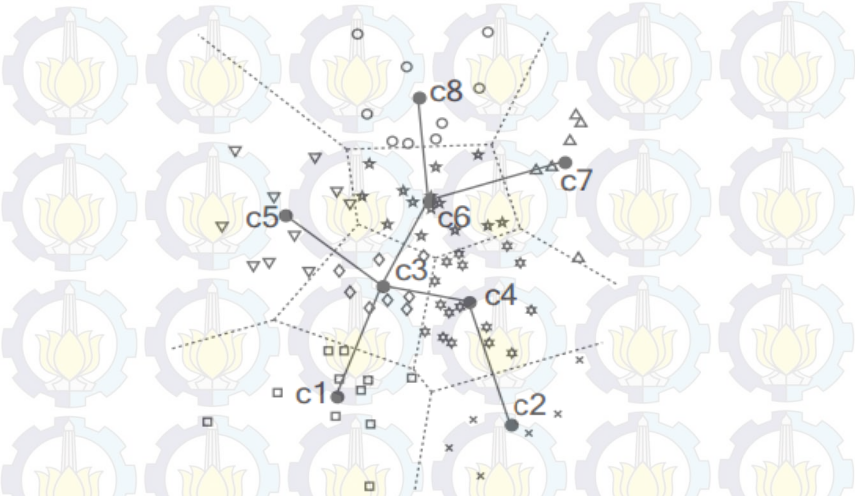
Gambar 2.20. Dataset yang terpartisi

Pada Gambar 2.21, suatu *MST* dibentuk dari *centroid* tiap *subset*. *Subset* yang *centroid* nya dihubungkan pada *MST centroid* merupakan *subset* yang bertetangga. *Subset c2* bertetangga dengan *subset c4*. *Subset c1* bertetangga dengan *subset c3.* Untuk setiap *subset* yang bertetangga dicari *edge* penghubungnya dengan tahap selanjutnya, *Detecting the Connecting Edge*. Hal ini ditunjukkan pada Gambar 2.22.

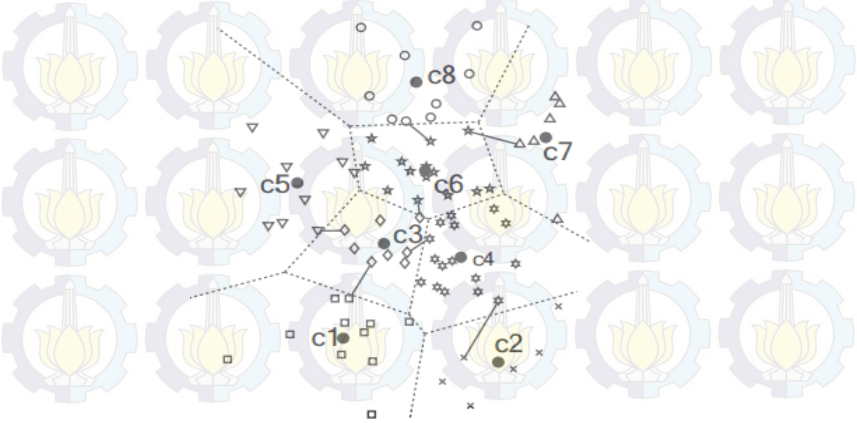
2.5.3 *Detecting the Connecting Edge*

Untuk setiap *subset* yang bertetangga, dicari *edge* yang menjadi penghubung antara dua *subset*. Cara yang digunakan adalah menghubungkan titik yang jaraknya paling dekat dengan *centroid subset* tetangga. Setelah semua *subset* terhubung, maka didapatkan satu *MST* perkiraan. Ilustrasi tahap ini ditunjukkan pada

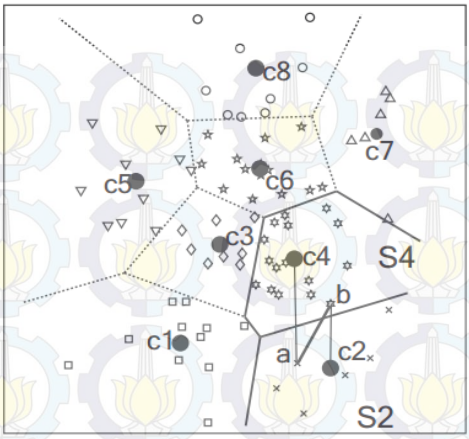
Gambar 2.23 [6]. Pada Gambar 2.23, ditunjukkan bahwa *vertex a* merupakan *vertex* didalam *subset S2* yang paling dekat dengan *centroid subset S4. Vertex b* merupakan *vertex* didalam *subset S4* yang paling dekat dengan *centroid subset S2.* Maka, *edge(a,b)* merupakan *edge* penghubung dari *subset S2* dan *S4.*



Gambar 2.21. MST centroid



Gambar 2.22. Subset yang bertetangga dicari edge penghubungnya

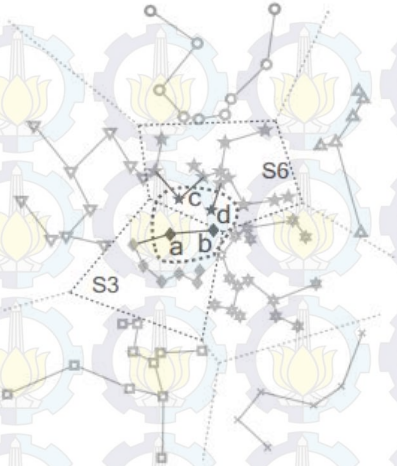


Gambar 2.23. Tahap Detecting the Connecting Edge

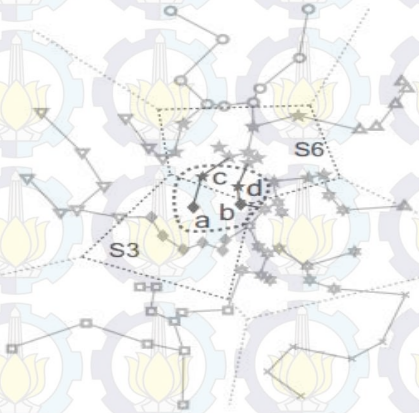
2.5.4 *Second Approximate MST*

Pada tahap ini, dilakukan pengulangan 3 tahap diatas, untuk memperbaiki hasil *MST* perkiraan pertama. Hasil *MST* perkiraan pertama sangat jauh bila dibandingkan dengan *MST* aslinya. Ini disebabkan pada saat *MST* per *subset* dibangun, data yang terletak pada perbatasan terbentuk di dalam *subset*, tidak melintasi *subset* yang lain. Pada Gambar 2.25 [6], dapat dilihat bahwa pada *MST* yang benar, *vertex* a menyambung dengan *vertex* c, dan *vertex* b menyambung dengan *vertex* d. Sedangkan pada *MST* perkiraan pertama yang ditunjukkan pada Gambar 2.24, *vertex* a menyambung dengan *vertex* b, *vertex* c menyambung dengan *vertex* d karena terdapat pada satu *subset*.

Data dibagi menjadi *k*-1 *subset* menggunakan algoritma klastering K-Means dengan cara menjadikan titik tengah dari *edge* - *edge* yang menjadi bagian dari *MST centroid* tiap *subset* menjadi *centroid* yang baru. Ilustrasi perhitunan *midpoint* ini ditunjukkan pada Gambar 2.26. Titik tengah ini biasanya terletak di sekitar perbatasan antar *subset*, sehingga dapat menyelesaikan permasalahan seperti yang ditunjukkan pada Gambar 2.24.

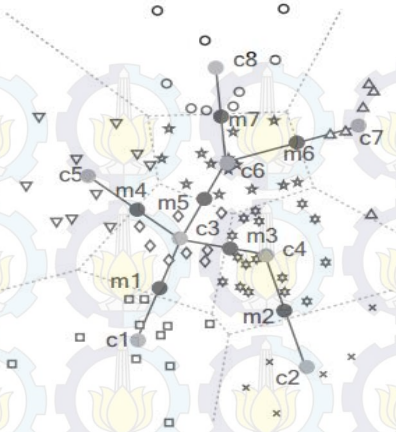


Gambar 2.24. Kasus Pembentukan MST yang Salah



Gambar 2.25. Ilustrasi MST yang sebenarnya

Ilustrasi *dataset* dipartisi menggunakan *centroid* yang baru ditunjukkan pada Gambar 2.27. Lalu step *Combine Subset Algorithm* dan *Detecting the Connecting Edge* dilakukan. Pada akhir tahap ini, didapatkan *MST* perkiraan yang kedua.



Gambar 2.26. Perhitungan midpoint dari MSTcentroid

Gambar 2.27. Partisi menggunakan centroid baru

2.5.5 *Merge Algorithm*

Pada tahap ini, dua *MST* perkiraan yang didapat akan digabung menjadi sebuah *graph*. *Graph* ini memiliki paling banyak

memiliki *edge* sejumlah 2(*N-1*). Kemudian, *graph* ini dicari *MST* nya menggunakan algoritma *MST* konvensional. Pada akhir tahap ini, diperoleh *MST* perkiraan terakhir.

2.6 Klastering

Klastering adalah proses mengelompokkan data berdasarkan kemiripan antar data. Data yang memiliki kemiripan diletakkan pada satu kelompok yang sama, sedangkan yang tidak memiliki kemiripan diletakkan pada kelompok yang berbeda.

2.6.1 *K-Means*

K-Means adalah salah satu algoritma klastering (pengelompokan) yang paling umum. Algoritma ini mengelompokkan objek ke beberapa klaster, berdasarkan kedekatan dengan *centroid* [16]. Algoritma ini membutuhkan parameter jumlah klaster yang ingin dibentuk. *K-Means* dimulai dengan pemilihan *centroid* (titik tengah), satu untuk masing-masing klaster. Pemilihan *centroid* bisa secara acak, ataupun ditentukan. Setelah itu, masing-masing data yang ingin dikelompokkan dihitung jaraknya dengan masing-masing *centroid.* Perhitungan jarak dilakukan menggunakan *Euclidean Distance* dengan rumus (2.1).

Setelah dilakukan perhitungan jarak, data akan dimasukan ke dalam klaster yang sama berdasarkan *centroid* yang paling dekat. Hal ini terus dilakukan sampai data tidak berpindah-pindah klaster. Tahapan algoritma *K-Means* ditunjukkan pada Gambar 2.28. Sebagai contoh pada Gambar 2.29 [17]. Pada *step* 1, ditentukan 2 titik dari data sebagai *centroid.* Kemudian tiap data dikelompokkan dengan *centroid* yang terdekat. Pada *step 2* ditunjukkan bahwa *centroid* baru dihitung dengan cara menghitung rata-rata dari semua data yang terletak pada suatu kelompok. Pada *step 3* ditunjukkan, data dikelompokkan kembali dengan *centroid* baru yang didapat pada *step 2. Step* 4 mengilustrasikan penghitungan *centroid* yang baru untuk digunakan dalam

pengelompokan data pada *step 5*. Pada *step 6*, dilihat bahwa data pada tiap klaster tidak berubah. Maka, proses klastering selesai.

$$d(P,Q) = \sqrt{\sum_{j=1}^{n} (Xj(P) - Xj(Q))^2} \quad (2.1)$$

d(P,Q) = jarak / nilai Euclidean Distance titik P ke Q

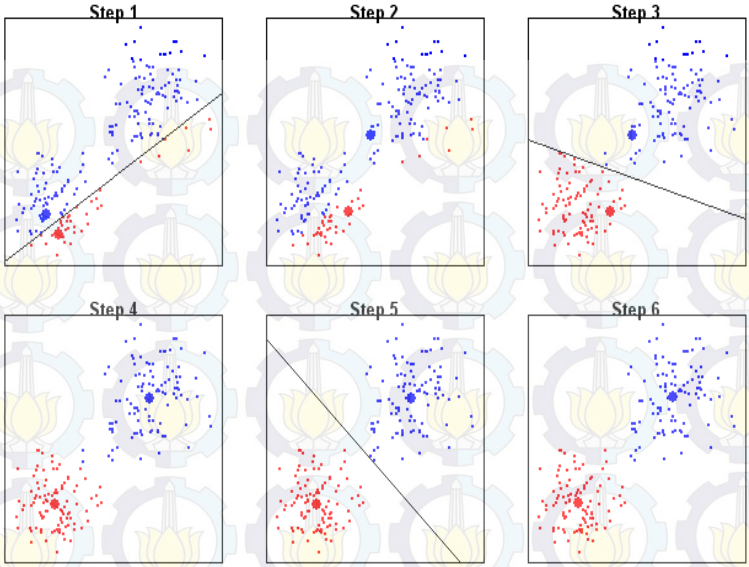
P = titik ke- 1 ; Q = titik ke-2

n = jumlah fitur

Xj (P) = nilai fitur ke j pada titik P

Masukan : *k*: Jumlah klaster
X: *Dataset* yang terdiri dari n objek
Keluaran : *k* klaster yang masing-masing berisi beberapa data
Metode :
(1). Tetapkan *k* titik dari *X* sebagai *centroid* (jika tidak ditentukan, maka pilih *k* titik secara *random*).
(2). Lakukan
Kelompokkan tiap data berdasarkan *centroid* yang paling dekat
Hitung *centroid* yang baru
(3). Sampai data dalam tiap klaster tidak berubah

Gambar 2.28. Tahapan Algoritma K-Means

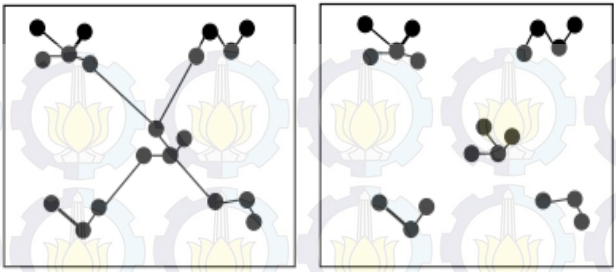


Gambar 2.29. Ilustrasi pembentukan klaster dengan K-Means

2.6.2 Klastering menggunakan *Minimum Spanning Tree*

Algoritma yang dipakai adalah algoritma yang diusulkan oleh Prasanta Jana dan Azad Naik. Algoritma ini ditujukan untuk melakukan pengelompokan data menggunakan informasi yang terdapat pada *MST*. Algoritma ini menggunakan konsep penghapusan *edge* yang melebihi *threshold*. *K edge* yang dihapus akan memisahkan *graph* menjadi *k+1 subgraph* yang terpisah dan dianggap merupakan klaster tersendiri.

Ilustrasi klastering menggunakan *MST* ditunjukkan pada Gambar 2.30 [7]. Pada gambar tersebut, dilakukan penghapusan 4 *edge* yang paling panjang, sehingga, terbentuk 5 klaster.



Gambar 2.30. Representasi MST dan klastering berdasarkan MST

2.7 Indeks Dunn

Indeks Dunn adalah suatu indeks yang dapat digunakan untuk menguji validitas klaster. Indeks ini merupakan rasio jarak terkecil antara dua titik yang terletak pada dua klaster yang berbeda dengan jarak terbesar antara dua titik yang terletak pada satu klaster [18]. Nilai indeks ini antara 0 sampai ∞. Semakin besar nilai indeks Dunn, maka hasil klastering dinyatakan bagus [19]. Rumus Indeks Dunn ditunjukkan pada rumus (2.2).

$$D = \min_{1 \le i \le n} \{ \min_{1 \le j \le n, i \ne j} \{ \frac{d(i,j)}{\max_{1 \le k \le n} d'(k)} \} \} \quad (2.2)$$

D = nilai indeks Dunn
d(i,j) = jarak *Euclidean distance* antara titik pada klaster i dan klaster j
d'(k) = jarak *Euclidean distance* antara titik pada klaster k
n = jumlah klaster

2.8 Indeks Ray&Turi

Indeks Ray&Turi adalah suatu indeks yang dapat digunakan untuk menguji validitas klaster. Indeksi ini diusulkan oleh Ray dan Turi [7]. Konsep indeks ini berdasarkan *compactness* dan *isolation*. *Compactness* adalah ukuran kohesi antar data, dan *isolation* adalah ukuran perbedaan antar klaster.

Compactness diukur dengan rata-rata jarak semua titik dalam satu klaster ke *centroid* klaster tersebut. Rumus ini ditunjukkan pada rumus (2.3).

$$\text{Intra} = \frac{1}{\text{N}} \sum_{\text{i}=1}^{\text{k}} \sum_{x \in Ci} \|x - zi\|^2 \quad (2.3)$$

Intra = nilai *compactness* / intra-klaster
N= jumlah data
k = jumlah klaster
z_i = *centroid* klaster ke *i*
C_i = klaster ke *i*
x = data

Sedangkan *isolation*, diukur dengan mencari jarak antar dua *centroid* klaster yang paling kecil. Rumus ini ditunjukkan pada rumus (2.4).

$$Inter = \min \left\| z_i - z_j \right\|^2 ; i \neq j \quad (2.4)$$

Inter = nilai *isolation* / inter-klaster
z_i = *centroid* klaster ke *i*
z_j = *centroid* klaster ke *j*

Rasio yang merupakan hasil dari *compactness* dibagi *isolation*, merupakan nilai yang digunakan untuk menguji validitas klaster. Semakin kecil nilai indeks ini, maka hasil klastering dinyatakan semakin baik. Rumus ini ditunjukkan pada rumus (2.5).

$$Ratio(Validity) = \frac{Intra}{Inter} \quad (2.5)$$

Ratio= nilai validitas indeks klaster
Intra = nilai *compactness* / intra-klaster
Inter = nilai *isolation* / inter-klaster

BAB III
DESAIN PERANGKAT LUNAK

Pada bab ini akan dijelaskan perancangan program yang dibuat. Perancangan akan dibagi menjadi dua proses utama, yaitu:

1. Desain *Fast Minimum Spanning Tree*.
2. Desain klastering berdasarkan *Minimum Spanning Tree.*

Pada bab ini akan dijelaskan gambaran umum setiap program utama dalam *pseudocode*.

3.1 Desain Metode Secara Umum

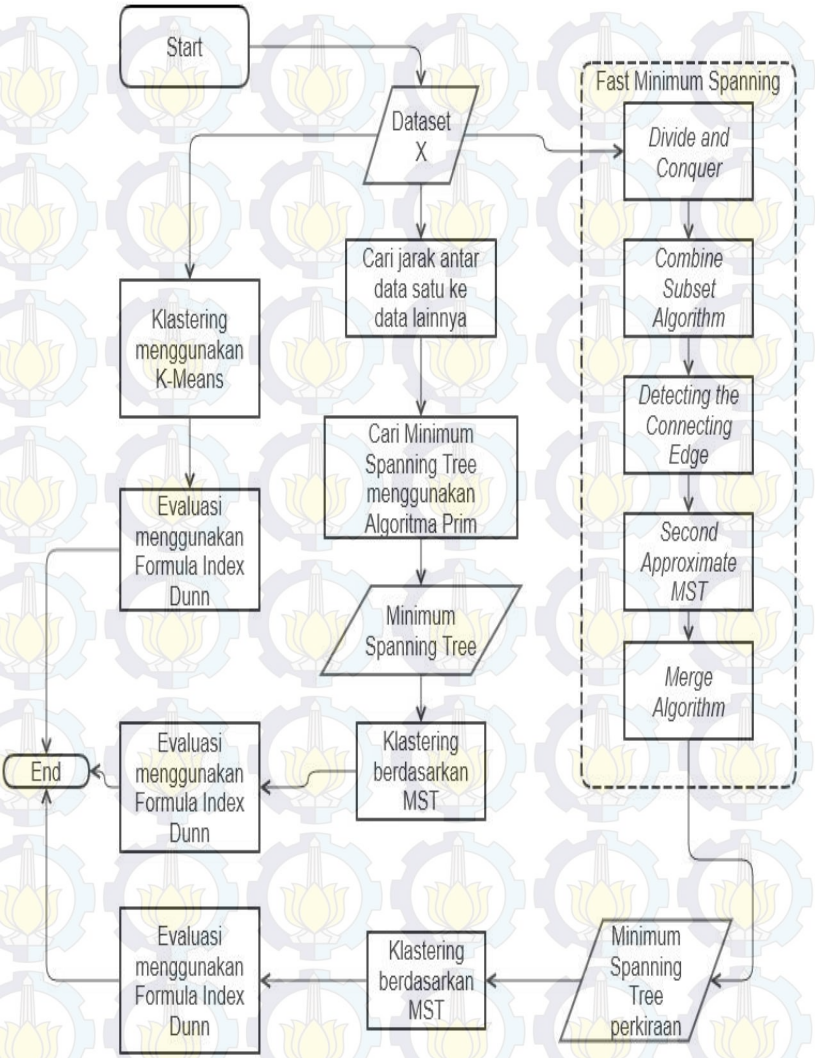
Pada tugas akhir ini akan dibangun suatu sistem yang dapat mengimplementasikan *Fast Minimum Spanning Tree (FMST)* dan klastering menggunakan *Minimum Spanning Tree (MST)*. Sistem akan menerima masukan *dataset* yang mengandung data bersifat numerik.

Dalam tugas akhir ini, proses pertama yang dilakukan adalah melakukan metode *FMST* terhadap *dataset* masukan. Kemudian, mencari jarak tiap satu data dengan data lainnya dengan rumus *Euclidean* distance. Kemudian, *dataset* tersebut dicari *MST* nya menggunakan metode algoritma Prim.

Klastering dilakukan menggunakan *MST* dari metode *FMST* dan algoritma Prim, serta menggunakan K-Means. Kemudian hasil klastering dari 3 metode tersebut diuji menggunakan rumus Indeks Dunn. Alur Program Utama terdapat pada Gambar 3.1.

3.2 Desain Algoritma Prim

Algoritma ini digunakan untuk menyelesaikan permasalahan *MST*. Algoritma Prim yang digunakan menggunakan struktur data *priority queue. Priority queue* tersebut didesain berdasarkan *low priority*, dimana data yang bernilai lebih kecil diprioritaskan. Desain Algoritma Prim terdapat pada Gambar 3.2.



Gambar 3.1. Alur Program Utama

Masukan	*Graph G = (V,E)*, himpunan *vertex V* dan himpunan *edge E*
Keluaran	*Minimum Spanning Tree* dari *G*

```
1. Tentukan s sebagai vertex awal; s ∈ V.
2. Set visited(s) = true
3. Masukkan semua edge yang memiliki vertex awal s
   beserta bobot edge tersebut kedalam priority
   queue
4. Selama priority queue tidak kosong,ulangi :
   Ambil puncak dari priority queue =>edge(s, x),w;
   s merupakan vertex awal, x merupakan vertex
   akhir, dan w merupakan bobot dari edge yang
   menghubungkan vertex s dan vertex x.
   Masukkan edge tersebut sebagai anggota himpunan
   dari MST.
   Hapus puncak dari priority queue.
   Jika visited(x) != true, lakukan langkah ke 5
5. Set visited(x) = true
   Masukkan semua edge yang memiliki vertex awal x
   dan visited(y)!= true, dimana y adalah vertex
   akhir, beserta bobot edge tersebut kedalam
   priority queue
6. Selesai
```

Gambar 3.2. Pseudocode Algoritma Prim

3.3 Desain Metode *Fast Minimum Spanning Tree*

Pada bagian ini dijelaskan tiap tahapan dari *Fast Minimum Spanning Tree* disertai *pseudocode*. *FMST* terdiri dari 5 tahap, *Divide and Conquer*, *Combine Subset Algorithm*, *Detecting the Connecting Edge*, *Second Approximate MST*, dan *Merge Algorithm*. Desain *FMST* secara keseluruhan dapat dilihat pada Gambar 3.3.

Masukan	*Dataset X* yang terdiri dari *N* data dan *m* fitur.
Keluaran	*Minimum Spanning Tree* perkiraan dari *dataset X*

1. Terapkan *Divide and Conquer* terhadap *X* untuk mendapatkan *k* buah *MST*
2. Terapkan *Combine Subset Algorithm* untuk mencari *subset* yang bertetangga dan *Detecting the Connecting Edge* untuk mendapatkan *edge* yang menggabungkan *subset* yang bertetangga. Tahap ini menghasilkan *MST* perkiraan pertama.
3. Terapkan Second Approximate *MST* untuk mendapatkan *MST* perkiraan kedua.
4. Gabung *MST* perkiraan pertama dan kedua untuk mendapatkan *graph G*.
5. Terapkan algoritma Prim pada *G* untuk mendapatkan *MST* perkiraan dari *X*.

Gambar 3.3. Pseudocode Metode Fast Minimum Spanning Tree

3.3.1 Tahap *Divide and Conquer*

Tahap *Divide and Conquer* adalah tahap pertama dalam metode *FMST*. *Dataset* dipartisi ke $\sqrt{N}$ *subset*, dimana *N* merupakan jumlah data dengan menggunakan algoritma klastering K-Means. Pada tiap *subset*, hitung jarak antar tiap data didalamnya dengan menggunakan *Euclidean distance*. Perhitungan dilakukan untuk setiap kombinasi pasangan data. Langkah ini menjadikan tiap *subset* seperti menjadi *complete graph* tersendiri dengan tiap data didalam *subset* menjadi *vertex* dan jarak antar tiap data menjadi bobot dari *edge* yang menghubungkan pasangan data. Tiap *subset* dicari *MST* nya dengan menggunakan algoritma Prim.

Masukan pada tahap ini adalah *dataset* yang ingin dicari *MST* nya. Keluaran pada tahap ini adalah $\sqrt{N}$ *MST* dari *subset*. Desain tahap *Divide and Conquer* terdapat pada Gambar 3.4.

Masukan	*Dataset X*, yang terdiri dari *N* data dan m fitur.
Keluaran	*k* buah *MST subset* => MST_i ; $1 \leq i \leq k$

1. Set k= $\sqrt{N}$
2. Terapkan algoritma klastering Kmeans dengan parameter *k* ke *dataset X* untuk mendapatkan sebanyak *k subset*. Simpan pada variabel $S=\{S_1,S_2,\ldots,S_k\}$
3. Untuk setiap *subset*, hitung jarak antar satu data dengan data lainnya dalam *subset* tersebut menggunakan *Euclidean Distance* untuk menghasilkan *k complete graph* $P=\{P_1,P_2,\ldots,P_k\}$; $1 \leq i \leq k$
4. Terapkan algoritma Prim pada setiap anggota dari *P* untuk mendapatkan *k MST* => MST_i ; $1 \leq i \leq k$

Gambar 3.4. Pseudocode Tahap Divide and Conquer

3.3.2 Tahap *Combine Subset Algorithm*

Tahap *Combine Subset Algorithm* adalah tahap untuk menggabungkan *k MST* yang telah dihasilkan pada tahap *Divide and Conquer*. Untuk menggabungkan *k MST*, diperlukan *edge* yang menghubungkan *subset* yang bertetangga. *Subset* yang bertetangga adalah *subset* yang *centroid* (titik pusat) nya dihubungkan pada *MSTcentroid. MSTcentroid* adalah *MST* dari *graph* yang terdiri dari *centroid* tiap *subset* sebagai *vertex*, serta *edge* yang menghubungkan tiap *centroid* tersebut. *Edge* penghubung *subset* yang bertetangga diselesaikan dengan cara pada tahap selanjutnya yaitu, *Detecting the Connecting Edge*. Setelah didapat *k-1 edge* penghubung, tambahkan *k-1 edge* tersebut dengan *k MST* untuk membentuk suatu himpunan yang merupakan *MST* perkiraan pertama. Desain tahap *Combine Subset Algorithm* dapat dilihat pada Gambar 3.5.

Masukan	*k* buah *MST subset* => MST_i ; $1 \leq i \leq k$
Keluaran	*MST* perkiraan pertama dari *dataset X* => *MST1* *MST* dari tiap *centroid subset* => *MSTcen*

1. Hitung *centroid subset* S_i => C_i; $1 \leq i \leq k$. *Centroid* merupakan titik tengah/ pusat dari data pada *subset* tersebut.
2. Hitung jarak tiap C_i *menggunakan Euclidean Distance* untuk membentuk *graph G*, dimana semua C_i menjadi anggota himpunan *vertex V* dan jarak tiap C_i menjadi anggota himpunan *edge E*.
3. Terapkan algoritma Prim pada *graph G* untuk mendapatkan *MSTcen*.
4. Untuk setiap *subset* yang *centroid* nya dihubungkan oleh *edge e*, dimana $e \in$ *MSTcen*, terapkan *Detecting Connecting Edge* untuk mendapatkan *edge* penghubung.
5. Gabung *k-1 edge* penghubung dengan *k MST* menjadi *MST1*.

Gambar 3.5. Pseudocode Combine Subset Algorithm

3.3.3 Tahap *Detecting the Connecting Edge*

Pada tahap ini, setiap pasangan *subset* yang dijadikan sebagai masukan, akan dicari *edge* yang menghubungkannya. *Edge* tersebut terdiri dari *vertex* dalam *subset* yang paling dekat dengan *centroid subset* pasangannya. Konsep jarak yang dipakai adalah *Euclidean distance*. Jika terdapat *k subset*, maka terdapat *k*-1 *edge* penghubung. Desain tahap *Detecting the Connecting Edge* terdapat pada Gambar 3.6.

3.3.4 Tahap *Second Approximate MST*

Tahap ini merupakan tahap untuk mencari *MST* perkiraan yang kedua. Cara yang digunakan sama dengan cara yang digunakan untuk mendapatkan *MST* perkiraan yang pertama.

Hanya saja, partisi yang digunakan berbeda. *MSTcen* yang didapat pada tahap *Connecting Subset Algorithm* dihitung titik tengahnya (*midpoint*). Tiap data dikelompokkan menggunakan algoritma K-Means dengan menginisialisasi *midpoint* sebagai *centroid* untuk membentuk *k-1 subset* yang baru. Lalu tiap *subset* tersebut dicari *MST* nya masing-masing dengan algoritma Prim. Kemudian, terapkan tahap *Combine Subset Algorithm* pada *k-1 subset* tersebut. Pada akhir tahap ini, didapatkan *MST* perkiraan yang kedua. Desain tahap *Second Approximate MST* terdapat pada Gambar 3.7.

Masukan	Pasangan *subset* yang akan dihubungkan => (S_i, S_j)
Keluaran	*Edge e* yang menghubungkan *subset* S_i dan *subset* S_j

1. Cari *a*, *vertex* pada *subset* S_i yang paling dekat dengan *centroid subset* S_j.
2. Cari *b*, *vertex* pada *subset Sj* yang paling dekat dengan *centroid subset* S_i.
3. *Edge e*, *edge* yang menghubungkan *vertex a*, dan *vertex b*, merupakan penghubung *subset* S_i dan S_j.

Gambar 3.6. Pseudocode Detecting the Connecting Edge

Masukan	*MSTcen*, *Dataset X* yang terdiri dari N data dan m fitur
Keluaran	*MST* perkiraan kedua dari *dataset X* => *MST*2

1. Hitung *midpoint* (titik tengah) untuk setiap *edge* $e \in MSTcen$.
2. Partisi *dataset X* dengan algoritma klastering K-Means menggunakan *midpoint* sebagai *centroid* menjadi *k-1 subset* $S'=\{S'_1, S'_2, \ldots, S'_{k-1}\}$.
3. Hitung jarak antar satu data dengan data lainnya menggunakan *Euclidean Distance* dalam *subset* S'i untuk menghasilkan *k-1* complete *graph* $P'=\{P'_1,P'_2,\ldots,P'_{k-1}\}$; $1 \leq i \leq k-1$.
4. Terapkan algoritma Prim pada tiap anggota *P'* untuk mendapatkan *k-1 MST'* => MST'_i; $1 \leq i \leq k-1$.
5. Terapkan *Combine Subset Algorithm* pada *k-1 MST'* untuk mendapatkan *MST* perkiraan kedua.

Gambar 3.7. Pseudocode Second Approximate MST

3.3.5 Tahap *Merge Algorithm*

Tahap ini adalah tahap paling akhir dalam metode *FMST*. *MST1* dan *MST2* yang merupakan *MST* perkiraan yang didapat pada tahap-tahap sebelumnya digabung menjadi sebuah *graph*. *Graph* ini memiliki tidak lebih dari 2(*N*-1) *edge.* Kemudian, algoritma Prim diterapkan pada *graph* tersebut untuk mendapatkan *MST* perkiraan terakhir. Desain tahap *Merge Algorithm* terdapat pada Gambar 3.8.Gambar 3.7

Masukan	*MST1, MST2*
Keluaran	*MST* perkiraan terakhir => *FMST*
1. Gabung *MST1* dan *MST2* menjadi sebuah *graph G*. 2. Terapkan algoritma Prim terhadap *G* untuk mendapatkan *MST* perkiraan terakhir => *FMST*.	

Gambar 3.8. Pseudocode Merge Algorithm

3.4 Desain Metode Klastering berdasarkan *Minimum Spanning Tree*

Metode ini adalah metode yang diusulkan oleh Prasanta Jana dan Azad Naik. Konsep pengelompokan dari metode ini adalah dengan cara menghapus *edge* yang ada pada *MST dataset*. Dengan menghapus *k edge*, maka akan menghasilkan *k+1* klaster. *Edge* yang dihapus merupakan *edge* yang terpanjang atau memiliki bobot melebihi *threshold* yang telah ditentukan. Hasil klastering yang diperoleh, dihitung rasio nya dengan rumus yang diusulkan Ray & Turi. Ulangi langkah-langkah diatas dengan menambah nilai *threshold.* Langkah- langkah tersebut diulangi sampai semua *edge* telah terhapus. Indikator klastering tersebut adalah yang paling baik, adalah ketika rasio yang didapatkan bernilai paling kecil. Desain Metode klastering berdasarkan *MST* terdapat pada Gambar 3.9.

Masukan	*Minimum Spanning Tree*, himpunan *edge* yang menghubungkan *vertex awal dan vertex akhir* dengan bobot *Euclidean Distance*, dari *dataset X*
Keluaran	*threshold*

1. Tentukan nilai *threshold*, dan *step_size.*
2. Tentukan nilai perulangan *i*=1.
3. Selama *threshold* < *edge* terpanjang dari *Minimum Spanning Tree*, ulangi :
   Hapus *edge* yang memiliki bobot lebih besar dari *threshold.* Untuk setiap *k edge* yang dihapus, maka tercipta *k-1* klaster.
   Evaluasi hasil klastering menggunakan uji validitas index Ray&Turi => *ratio(i).*
   threshold = threshold + step_size.
   i=i+1.
4. Mengembalikan nilai threshold pada perulangan ke *i,* dimana *ratio(i)* merupakan nilai terkecil dibanding *ratio* lainnya.

Gambar 3.9. Pseudocode klastering menggunakan MST

3.5 Desain Metode Indeks Dunn

Indeks Dunn merupakan suatu indeks yang dapat digunakan untuk menguji validitas klaster. Desain metode ini dapat dilihat pada. Desain metode Indeks Dunn dapat dilihat pada Gambar 3.10.

Masukan	*Dataset X*, yang terdiri dari m data dan n+1 fitur. Data uji coba memiliki n=2 fitur atau dimensi. Fitur terakhir berisi label klaster setiap data
Keluaran	Nilai uji evaluasi Indeks Dunn

1. Cari nilai *a* yang merupakan jarak terpendek antar data berdasarkan *Euclidean Distance* dalam klaster berbeda. Pencarian dilakukan pada semua kombinasi pasangan data.
2. Cari nilai *b* yang merupakan jarak terpanjang antar data berdasarkan *Euclidean Distance* dalam klaster yang sama. Pencarian dilakukan pada semua kombinasi pasangan data.
3. Mengembalikan nilai uji evaluasi = *a/b* sebagai keluaran.

Gambar 3.10. Pseudocode Metode Indeks Dunn

[Halaman ini sengaja dikosongkan]

BAB IV
IMPLEMENTASI

Pada bab ini akan dibahas mengenai implementasi yang dilakukan berdasarkan rancangan yang telah dijabarkan pada bab sebelumnya. Sebelum penjelasan implementasi akan ditunjukkan terlebih dahulu lingkungan untuk melakukan implementasi.

4.1 Lingkungan Implementasi

Lingkungan implementasi yang akan digunakan untuk melakukan implementasi adalah MATLAB yang diinstal pada sistem operasi *Windows* 8.1.

4.2 Implementasi

Pada subbab ini akan dijelaskan implementasi setiap subbab yang terdapat pada bab sebelumnya yaitu bab perancangan program. Pada bagian implementasi ini juga akan dijelaskan mengenai fungsi-fungsi yang digunakan dalam program tugas akhir ini dan disertai dengan kode sumber masing-masing fungsi utama.

4.2.1 Implementasi Tahap *Divide and Conquer*

Tahap *Divide and Conquer* adalah tahap pertama pada *Fast Minimum Spanning Tree (FMST)*, dimana *dataset* dibagi menjadi beberapa *subset* lalu diselesaikan *MST* nya per *subset*. *Dataset* dibagi menggunakan metode K-Means. Metode K-Means dapat dipanggil secara langsung karena disediakan sebagai *library* oleh MATLAB. *Dataset* dibagi menjadi √N *subset* dengan menggunakan *Euclidean distance*, dimana *centroid* merupakan rata-rata semua titik dalam satu *subset*.

Implementasi Tahap *Divide and Conquer* dapat dilihat pada Kode Sumber 4.1. Fungsi tersebut diimplementasikan pada

baris 1 dan baris ke 2. Pada baris ke 3 hingga baris ke 19 dijelaskan untuk setiap *subset*, dilakukan implementasi algoritma Prim untuk menyelesaikan permasalahan *MST* nya. Lalu hasil *MST* dari tiap *subset* ditampung di variable *mst1*.

```
1  Cluster  =floor(sqrt(size(X,1)));
2   [Index, Centroid] =
   kmeans(Data,Cluster,'distance','sqEuclidean,'
   Replicates',Cluster);
3  for i=1:size(Centroid,1)
4      temp=[]; count=1;
5      for j=1:size(Data,1)
6          if Index(j)== i
7              temp(count,:)= Data(j,:);
8              count=count+1;
9          end
10     end
11     if(size(temp,1)>1)
12         [mst,cost] = prim(temp);
13         plus = repmat(counter-1,size(mst));
14         mst1 = [mst1;mst+plus];
15     end
16     temp(:,end+1)= i;
17     sorted(counter:counter+count-2,:)        =
   temp();
18     counter=counter+count-1;
19 end
```

Kode Sumber 4.1. Kode Sumber Tahap Divide and Conquer

4.2.2 Implementasi Tahap *Combine Subset Algorithm*

Tahap *Combine Subset Algorithm* adalah tahap dimana *MST* dari tiap *subset* digabungkan. *Subset* yang digabungkan merupakan *subset* yang bertetangga. *Subset* yang bertetangga adalah *subset* yang *centroid* nya dihubungkan oleh *MSTcentroid*.

Implementasi Tahap *Combine Subset Algorithm* dapat dilihat pada Kode Sumber 4.2.

```
1  mstCentroid,costCentroid] = prim(Centroid);
```

Kode Sumber 4.2. Kode Sumber Tahap Combine Subset Algorithm

4.2.3 Implementasi Tahap *Detecting the Connecting Edge*

Pada tahap ini, dicari *edge* yang menghubungkan tiap *subset* yang bertetangga. Implementasi tahap *Detecting the Connecting Edge* dapat dilihat pada kode sumber Kode Sumber 4.3. *Edge* tersebut dibentuk oleh titik yang memiliki jarak paling kecil dengan *centroid* dari *subset* tetangga nya. Langkah ini dilakukan pada baris ke 2 hingga ke 17. Selanjutnya *edge* - *edge* tersebut digabungkan dengan *mst1* sebagai himpunan dari *MST* perkiraan pertama. . Langkah ini dilakukan pada baris ke 18.

```
1  edge=[];
2  for i=1:size(mstCentroid,1)
3      for j=1:size(mstCentroid,2)
4          minimum = Inf;
5          for k=1:size(sorted,1)
6              if sorted(k,end) ==
   mstCentroid(i,j)
7                  x =
   norm(Centroid(mstCentroid(i,end-j+1),:) -
   sorted(k,1:end-1));
8                  if x < minimum
9                      minimum = x;
10                     edge(i,j) = k;
11                 end
12             elseif sorted(k,end) >
   mstCentroid(i,j)
13                 break;
14             end
15         end
16     end
17 end
18 mst1 = [mst1;edge];
```

Kode Sumber 4.3. Kode Sumber Tahap Detecting the Connecting Edge

4.2.4 Implementasi Tahap *Second Approximate MST*

Tahap ini digunakan untuk mendapatkan *MST* perkiraan kedua. Tahap ini dilakukan sebagai tahap untuk memperbaiki *MST* perkiraan pertama yang hasilnya jauh dari *MST* aslinya akibat data yang terletak pada perbatasan tidak terselesaikan dengan baik.

Implementasi tahap ini hampir sama dengan implementasi untuk mendapatkan *MST* perkiraan pertama, dimana membutuhkan tahap *Divide and Conquer*, *Combine Subset Algorithm*, dan *Detecting the Connecting Edge*. Perbedaannya terletak pada partisi *dataset*. *Dataset* yang terdiri dari *N* data dibagi menjadi $\sqrt{N-1}$ *subset* berdasarkan kedekatan titik dengan *centroid* yang merupakan titik tengah dari *edge-edge* pada *MSTcentroid*. Implementasi untuk perhitungan *centroid* yang baru dapat dilihat pada Kode Sumber 4.4. Kode Sumber 4.5, dan Kode Sumber 4.6 merupakan implementasi *Second Approximate MST* yang memuat 3 tahap diatasnya, *Divide and Conquer* (baris 3-20), *Combine Subset Algorithm* (baris 31), dan *Detecting the Connecting Edge* (baris 32-55).

```
1  newCentroid = [];
2  for i=1:size(mstCentroid,1)
3      newCentroid(i,:) =
   (Centroid(mstCentroid(i,1),:) +
   Centroid(mstCentroid(i,2),:))/2;
4  end
5  sorted(:,end+1)=0;
6  for i=1:size(sorted,1)
7      minimum = Inf;
8      for j=1:size(newCentroid,1)
9          x = norm(sorted(i,1:size(Data,2))-
   newCentroid(j,:));
10             if x<minimum
11                 minimum=x;
12                 sorted(i,end) = j;
13             end
14     end
15 end
```

Kode Sumber 4.4. Kode Sumber Tahap Second Approximate MST

```
1  count=1;counter=1;newsorted=[];mst2=[];no=[];
2  %Build mst per subset
3  for i=1:size(newCentroid,1)
4      temp=[];
5      count=1;
6      for j=1:size(sorted,1)
7          if sorted(j,end)== i
8              temp(count,:)=
   sorted(j,1:size(Data,2));
9              no = [no;j];
10             count=count+1;
11         end
12     if(size(temp,1)>1)
13         [mst,cost] = prim(temp);
14         plus = repmat(counter-1,size(mst));
15         mst2=[mst2;mst+plus];
16     end
17     temp(:,end+1)= i;
18     newsorted(counter:counter+count-2,:) =
   temp();
19     counter=counter+count-1;
20 end
21 %hitung new centroid
22 newCentroidx=[];
23 newnoCluster=[];
24 for i=1:size(newCentroid,1)
25     if size(find(newsorted(:,end)==i),1)>0
26         newCentroidx=
   [newCentroidx;newCentroid(i,:)];
27         newnoCluster = [newnoCluster;i];
28     end
29 end
30 %Combine Subset
31 [mstnewCentroid,costnewCentroid] =
   prim(newCentroidx);
32 %Detecting the Connecting Edge
33 ansx=zeros(1,size(mstnewCentroid,1)*
   size(mstnewCentroid,2));
34 for i =1 :
   size(mstnewCentroid,1)*size(mstnewCentroid,2)
```

Kode Sumber 4.5. Kode Sumber Tahap Second Approximate MST (2)

```
35      ansx(i) = newnoCluster(mstnewCentroid(i));
36  end
37  mstnewCentroid =
    reshape(ansx,[size(ansx,2)/2 , 2]);
38  edge=[];
39  for i=1:size(mstnewCentroid,1)
40      for j=1:size(mstnewCentroid,2)
41          minimum = Inf;
42          for k=1:size(newsorted,1)
43              if newsorted(k,end) ==
    mstnewCentroid(i,j)
44                  x =
    norm(newCentroid(mstnewCentroid(i,end-j+1),:)
    - newsorted(k,1:end-1));
45                  if x < minimum
46                      minimum = x;
47                      edge(i,j) = k;
48                  end
49              elseif newsorted(k,end) >
    mstnewCentroid(i,j)
50                      break;
51              end
52          end
53      end
54  end
55  mst2 = [mst2;edge];
56  % hold on
57
58  mst2adjust=[];
59  for i =1 : (size(mst2,1)*size(mst2,2))
60      mst2adjust(i)= no(mst2(i));
61  end
62  mst2adjust =
    reshape(mst2adjust,[size(mst2adjust,2)/2 ,
    2]);
63
64  mst2 = sort(mst2adjust')';
65  mst1= sort(mst1')';
66
```

Kode Sumber 4.6. Kode Sumber Tahap Second Approximate MST (3)

4.2.5 Implementasi Tahap *Merge Algorithm*

Implementasi tahap ini ditunjukkan pada Kode Sumber 4.7. Pada tahap ini, *MST* perkiraan pertama dan kedua digabung menjadi satu *graph* (pada baris 2-3). Lalu algoritma Prim diterapkan pada *graph* tersebut untuk mendapatkan *MST* perkiraan terakhir.

```
1  G = [mst1;mst2];
2  G = unique(G,'rows');
3  x = unique(G(:,1:2));
4  [mstfinal costfinal] =
   primadjlistedit(G,size(Data,1));
```

Kode Sumber 4.7. Kode Sumber Tahap Merge Algorithm

4.2.6 Implementasi Algoritma Prim

Algortima Prim digunakan untuk menyelesaikan permasalah *MST*. Untuk implementasi ini membutuhkan struktur data *adjacency list* dan *min weight priority queue*. List digunakan untuk menyimpan informasi titik tetangga dan *weight*. Sedangkan *priority queue* diimplementasikan berdasarkan *weight* yang paling bernilai minimum. Implementasi ini dapat dilihat pada Kode Sumber 4.8 dan Kode Sumber 4.9.

```
1  function [mst , jumlah] = prim(data)
2  PV = pdist(data,'euclidean');
3  a=list();
4  visited=zeros(1,size(data,1));
5  for i=1:size(data,1)
6      a.insert(i,list());
7  end
8  counter=1;
9  for i = 1 : size(data,1)-1
10     for j = i+1 : size(data,1)
11         weight = PV(counter);
12         a.getValue(i).insert(j,weight);
13         a.getValue(j).insert(i,weight);
14         counter = counter+1;
```

Kode Sumber 4.8. Kode Sumber Algoritma Prim

```
15      end
16  end
17  % tic
18  start = a.peek();
19  pqx=[];
20  for i=1:a.getValue(start).size
21      [c d] = a.getValue(start).get(i);
22      pqx = [pqx;d a.get(start) c];
23  end
24  pqx = sortrows(pqx,1);
25  visited(1,start)=1;
26  mst=zeros(size(data,1)-1,3);
27  counter=1;
28
29  while(size(pqx,1)>0)
30      s = pqx(1,1);
31      t = pqx(1,2);
32      u = pqx(1,3);
33      pqx = pqx(2:end,:);
34      if visited(1,u) ==0
35          visited(1,u) =1;
36          mst(counter,:) = [t,u,s];
37          counter = counter+1;
38          for i=1:a.getValue(u).size
39              [c d] = a.getValue(u).get(i);
40              if visited(1,c)==0
41                  pqx = [pqx;d a.get(u) c];
42              end
43          end
44          pqx = sortrows(pqx,1);
45      end
46  end
47
48  jumlah = sum(mst(:,3));
49
50  mst = mst(:,1:2);
```

Kode Sumber 4.9. Kode Sumber Algoritma Prim (2)

4.2.7 Implementasi Klastering berdasarkan *Minimum Spanning Tree*

Klastering berdasarkan *MST* menggunakan cara penghapusan *edge* pada *MST* yang memiliki bobot melebihi *threshold*. Setelah itu dihitung rasio antara *intra* klaster dan *inter* klaster. Hal tersebut diulangi terus menerus dengan memperbarui *threshold* sampai tidak ada lagi *edge* yang dihapus. *Threshold* diisi dengan nilai persentase, karena tiap *dataset* memiliki nilai rentang yang berbeda. Implementasi metode ini ditunjukkan pada Kode Sumber 4.10, Kode Sumber 4.11, Kode Sumber 4.12, dan Kode Sumber 4.13.

```
1  function[C thresholdx l ratio] =
   main_driver(mst,data,threshold,step_size)
2  [N,d]=size(data); % N,d holds the dimension of
   data
3  Inter=0.0;Intra=0.0;count=1;it=1;T=zeros(N,N)
   ;signal=0;
4  sorted=data;
5  for i=1:size(mst,1)
6          A(mst(i,1),mst(i,2))=1;
7          A(mst(i,2),mst(i,1))=1;
8  end
9  xe = pdist(data,'euclidean');
10 x = squareform(xe);
11 thresholdx=[];
12 rentang = max(mst(:,3))- min(mst(:,3));
13 step_size = rentang * step_size +
   min(mst(:,3));
14 threshold = rentang * threshold +
   min(mst(:,3));
15 while(signal == 0)
16 index=zeros(N,1);
17     counter1=1;
18     T=A; Cluster_no=0.0;
19     % storing end points with edge weight >
   threshold and removing that edge from MST
20     Storage=zeros(2*(N-1),d+1);
21     counter=0;
```

Kode Sumber 4.10. Kode Sumber klastering berdasarkan MST

```
for i=1:N-1
        for j=i+1:N
            if(T(i,j) == 1)
                if(sqrt(sum((sorted(i,:)-sorted(j,:)).^2)) < threshold)
                    counter = counter+1;
                    E(counter,:) = [i j];
                else
                    T(i,j)=0;T(j,i)=0;
Storage(counter1,:)=[sorted(i,:) i];
Storage(counter1+1,:)=[sorted(j,:) j];
                    counter1=counter1+2;
                end
            end
        end
    end
Cluster_no=(counter1+1) / 2; % Cluster_no stores the number of cluster center
    counter1=1;
    if(Storage(counter1,d+1) > 0) % more then one cluster
        % Finding Index of each sorted point
        counterx = 1;
        index=zeros(N,1);
        for i=1:size(E,1)
            if E(i,1)~=0
                if    index(E(i,1))==0    && index(E(i,2))==0
                    index(E(i,1))=counterx;
                    index(E(i,2))=counterx;
                    counterx=counterx+1;
                elseif index(E(i,1))==0
index(E(i,1))=index(E(i,2));
                elseif index(E(i,2))==0
index(E(i,2))=index(E(i,1));
                else
```

Kode Sumber 4.11. Kode Sumber klastering berdasarkan MST (2)

```
1                       if    index(E(i,1))    > index(E(i,2))
2                           index(E(i,1))      = index(E(i,2));
3                       elseif  index(E(i,2))  > index(E(i,1))
4                           index(E(i,2))      = index(E(i,1));
5                       end
6                   end
7               end
8           end
9           for i=1:size(E,1)
10              index(E(i,1))= min(index(E(i,1)),index(E(i,2)));
11              index(E(i,2))= min(index(E(i,1)),index(E(i,2)));
12          end
13          for i=1:N
14              if(index(i)==0)
15                  index(i)=counterx;
16                  counterx = counterx+1;
17              end
18          end
19          while(          Cluster_no          ~= size(unique(index)))
20              for i=1:size(E,1)
21                  index(E(i,1))= min(index(E(i,1)),index(E(i,2)));
22                  index(E(i,2))= min(index(E(i,1)),index(E(i,2)));
23              end
24          end
25          cluster =unique(index);
26
xc=zeros(Cluster_no,d);ff=0;summation=0;    % xc stores the cluster center
27          for i = 1:Cluster_no
28              Ind = find(index == cluster(i));
29              if(length(Ind) == 1)
```

Kode Sumber 4.12. Kode Sumber klastering berdasarkan MST (3)

```
                summation=summation + (max(xe))^2;
                %summation=summation + (max(PV(:,3)))^2;
            end
        end
        Inter=(min(pdist(xc)))^2;
        D=zeros(Cluster_no,N);
for i=1:N
            for k=1:Cluster_no
                D(k,i) = (sum(xc(k,:)-sorted(i,:)).^2);
            end
        end
        Intra=(sum(min(D)) + summation) / N;
        ratio(1,it)=Intra / Inter;

        C(1,it) = Cluster_no;
        thresholdx(1,it)=threshold;
        it=it+1;
        threshold=threshold + step_size;
    else % only one cluster is there
        Intra=0.0;xc=zeros(1,d);
        xc(1,:)=mean(sorted(:,:));
        for i=1:N
            Intra=Intra+(sum(xc(1,:)-sorted(i,:)).^2);
        end
        Intra=Intra/N;
        Inter=min(xe);
%         Inter=min(PV(:,3));
        if(Inter == 0)
            Inter=0.1;
        end
        C(1,it) = Cluster_no;
        ratio(1,it)=Intra / Inter;
        thresholdx(1,it)=threshold;
        signal =1;
    end
end
```

Kode Sumber 4.13. Kode Sumber klastering berdasarkan MST (4)

4.2.8 Implementasi Evaluasi Index Dunn

Index Dunn merupakan salah satu evaluasi internal hasil klastering. Evaluasi ini ditujukan untuk mencari agar klaster-klaster yang terbentuk memiliki kemiripan yang tinggi dalam satu klaster dan kemiripan yang rendah dalam klaster yang berbeda. Indeks ini merupakan rasio dari jarak terpendek titik beda klaster dengan jarak terpanjang titik yang terletak pada satu klaster. Semakin besar indeks Dunn yang dihasilkan, maka hasil klaster dinyatakan semakin baik. Implementasi Indeks Dunn ditunjukkan pada Kode Sumber 4.14.

```
1   function [DI num
    dem]=dunnsM(clusters_number,distM,ind)
2   i=clusters_number;
3   denominator=[];
4   cluster=unique(ind);
5   for i2=1:i
6       indi=find(ind==cluster(i2));
7       indj=find(ind~=cluster(i2));
8       x=indi;
9       y=indj;
10      temp=distM(x,y);
11      denominator=[denominator;temp(:)];
12  end
13  num=min(min(denominator));
14  neg_obs=zeros(size(distM,1),size(distM,2));
15
16  for ix=1:i
17      indxs=find(ind==cluster(ix));
18      neg_obs(indxs,indxs)=1;
19  end
20
21  dem=neg_obs.*distM;
22  dem=max(max(dem));
23
24  DI=num/dem;
25  end
```

Kode Sumber 4.14. Kode Sumber Indeks Dunn

[Halaman ini sengaja dikosongkan]

BAB V
UJI COBA DAN EVALUASI

Pada bab ini akan dijelaskan uji coba yang dilakukan pada aplikasi yang telah dikerjakan serta analisa dari uji coba yang telah dilakukan. Pembahasan pengujian meliputi lingkungan uji coba, skenario uji coba yang meliputi uji kebenaran dan uji kinerja serta analisa setiap pengujian.

5.1 Lingkungan Uji Coba

Lingkungan uji coba menjelaskan lingkungan yang digunakan untuk menguji implementasi *Fast Minimum Spanning Tree (FMST)*, dan klastering berdasarkan *Minimum Spanning Tree (MST)* pada Tugas Akhir ini. Lingkungan uji coba meliputi perangkat keras dan perangkat lunak yang dijelaskan sebagai berikut:

1. Perangkat keras
   a. Prosesor: Intel® Core ™ i5-2410M CPU @ 2.30GHz (4 CPUs), ~2.3GHz
   b. *Memory*(RAM): 4,00 GB
   c. Tipe sistem: 64-bit sistem operasi
2. Perangkat lunak
   a. Sistem operasi: *Windows* 8.1. Pro
   b. Perangkat pengembang: *MATLAB 2008*.

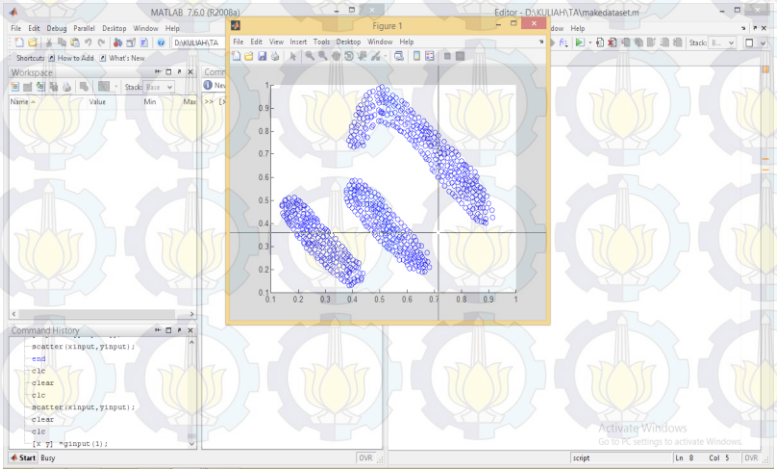
5.2 Data Uji Coba

Data yang digunakan merupakan data sintesis dan data real. Data sintesis adalah data yang didesain menyerupai situasi tertentu, yang mungkin tidak ditemui dalam data real. Dalam Tugas akhir ini, akan dibentuk data sintesis yang menyerupai bentuk-bentuk tertentu yang bersifat *non-Convex*.

Data-data ini dibuat secara manual oleh penulis. Data dibuat menggunakan *MATLAB* 2008 memanfaatkan fungsi yang sudah tersedia, yaitu *ginput()*. Perintah *ginput()* merupakan cara untuk memilih titik-titik dari grafik aktif dengan bantuan *mouse*. *[x,y]=ginput(n)* merupakan fungsi yang mengambil n titik dari sumbu aktif dan mengisikan koordinatnya dalam *array* kolom x dan y [20]. Contoh Kode Sumber untuk membuat data sintesis dapat dilihat pada Kode Sumber 5.1. Proses pembuatan *dataset* dapat dilihat pada Gambar 5.1.

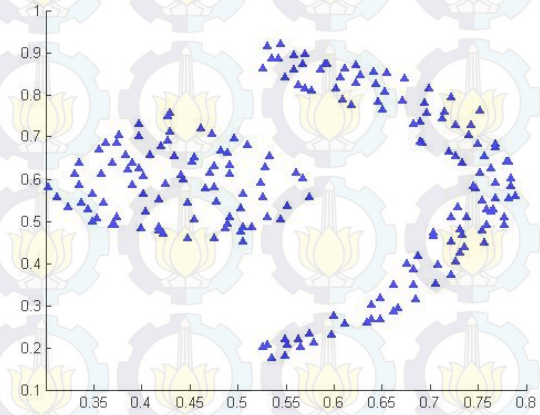
```
1.  ylim([0 1]);
2.  xlim([0 1]);
3.  xinput=[];
4.  yinput=[];
5.  scatter(xinput,yinput);
6.
7.  for i=1:50
8.      [x y] =ginput(1);
9.      xinput = [xinput x];
10.     yinput = [yinput y];
11.     scatter(xinput,yinput);
12. end
```

Kode Sumber 5.1. Kode Sumber membuat Data Sintesis

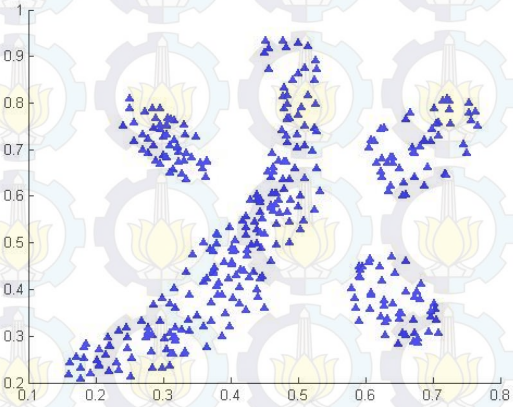


Gambar 5.1. Proses pembuatan data sintesis

Visualisasi data sintesis ini dapat dilihat pada Gambar 5.2, dan Gambar 5.3. Visualisasi data lebih lengkap dapat dilihat pada. LAMPIRAN. Data real yang dipakai merupakan *dataset iris* dan *wine* [21].



Gambar 5.2. Visualisasi Dataset T1



Gambar 5.3. Visualisasi Dataset T3

5.3 Skenario dan Evaluasi Pengujian

Uji coba ini dilakukan untuk menguji apakah fungsionalitas program telah diimplementasikan dengan benar dan berjalan sebagaimana mestinya. Uji coba akan didasarkan pada beberapa skenario untuk menguji kesesuaian dan kinerja aplikasi.

Pengujian terdiri dari 3 pengujian yaitu:

1. Perhitungan waktu dalam pembuatan *FMST dan MST*.
2. Perhitungan *weight error* yang didapat dalam pembuatan *FMST*.
3. Uji hasil klastering yang diperoleh dari klastering berdasarkan *FMST*, *MST*, dan K-Means.

5.3.1 Skenario Uji Coba dengan Data Sintesis

Uji coba dilakukan sebanyak 15 kali yang terdiri dari 15 data sintesis yang berbeda. Detil tiap percobaan dapat dilihat pada Tabel 5.1. *FMST* dilakukan 5 kali pada tiap data. Pada Gambar 5.4 ditunjukkan bahwa garis hijau adalah waktu rata-rata yang diperlukan metode *FMST* pada 5 kali percobaan. Sedangkan garis biru adalah waktu yang diperlukan untuk metode *FMST* dengan hasil yang terbaik pada 5 kali percobaan.

Uji coba juga dilakukan dengan menggunakan algoritma konvensional untuk menyelesaikan *MST*. Hasil waktu yang diperlukan oleh algoritma ini dapat dilihat pada Gambar 5.5.

Setelah mendapatkan hasil *weight* yang merupakan jumlah dari *edge-edge* yang diperoleh dari *FMST* dan algoritma konvensional *MST*, maka dilakukan perhitungan *weight error* yang menggunakan rumus:

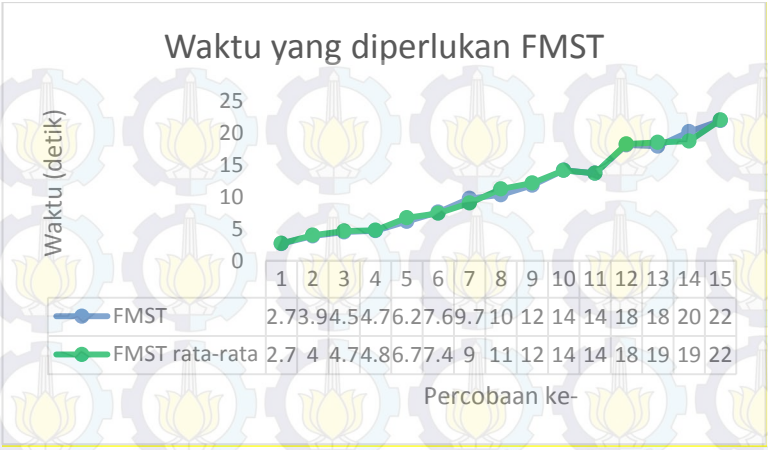
$$Weight\ error = \frac{weight(fmst) - weight(mst)}{weight(mst)} \quad (5.1)$$

Weight error = nilai *weight error*
Weight(fmst) = jumlah dari bobot *edge* pada *MST* yang dihasilkan *FMST*
Weight(mst) = jumlah dari bobot *edge* pada *MST* yang dihasilkan *MST* sebenarnya

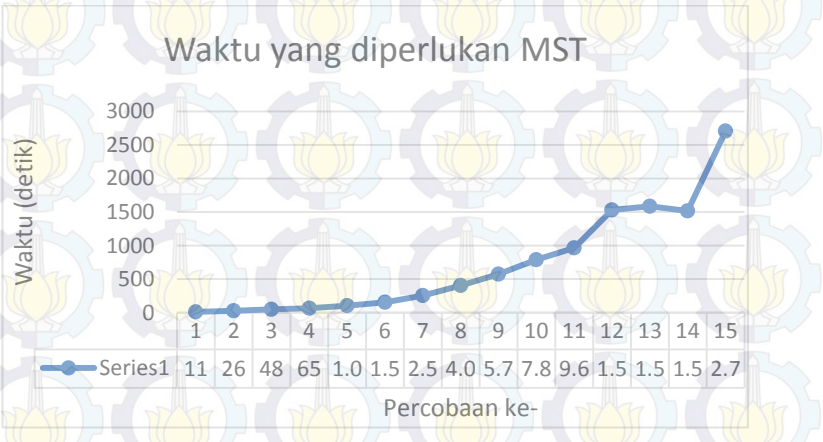
MST yang dihasilkan metode *FMST* dinilai baik jika memiliki *weight* yang sama dengan *MST* aslinya. Maka dari itu, semakin kecil *weight error*, maka *MST* yang dihasilkan metode *FMST* dinilai semakin baik.

Tabel 5.1. Tabel Detail Data Sintesis tiap Percobaan

Percobaan ke-	*Dataset* yang dipakai	Ukuran data (jumlah x fitur)	Jumlah klaster
1	T1	200 x 2	2
2	T2	250 x 2	3
3	Curve	300 x 2	2
4	Spiral1	312 x 2	3
5	T3	350 x 2	4
6	T4	400 x 2	5
7	T5	450 x 2	5
8	T6	500 x 2	5
9	T7	550 x 2	5
10	T8	600 x 2	2
11	Spiral2	612 x 2	3
12	T9	700 x 2	2
13	T10	700 x 2	3
14	T11	700 x 2	3
15	T12	800 x 2	4



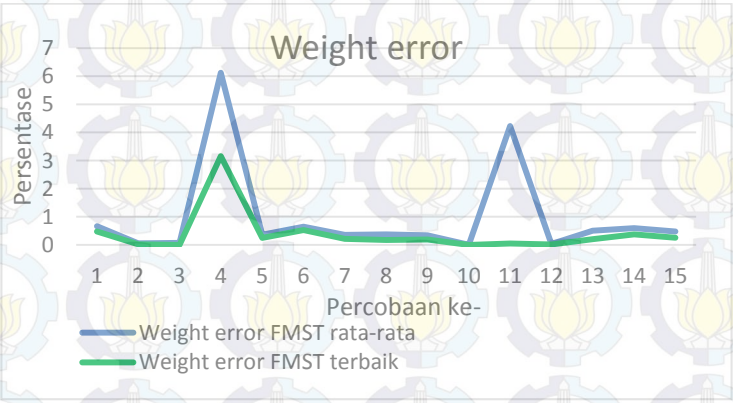
Gambar 5.4. Grafik Waktu yang diperlukan FMST



Gambar 5.5. Grafik Waktu yang diperlukan algoritma Prim

Hasil perhitungan *weight error* dapat dilihat pada Gambar 5.6. Pada Gambar 5.6 ditunjukkan bahwa garis biru adalah *weight error* rata-rata yang dihasilkan metode *FMST* pada 5 kali

percobaan. Sedangkan garis hijau adalah *weight error* terbaik yang dihasilkan untuk metode *FMST* dalam 5 kali percobaan.



Gambar 5.6. Grafik weight error

Setelah itu, dilakukan klastering berdasarkan *MST* yang didapat dari metode *FMST* dan algoritma konvensional *MST*, dan juga menggunakan algoritma klastering K-Means. Parameter jumlah klaster yang dipakai algoritma K-Means adalah jumlah klaster yang dihasilkan oleh klastering berdasarkan algoritma konvensional *MST*. Hasil klastering yang diperoleh dievaluasi menggunakan Indeks Dunn. Skenario Uji Coba dilakukan dengan *threshold* bernilai 0.1 dan *step_size* bernilai 0.1, 0.05, dan 0.025. Setelah dilakukan uji coba, ternyata hasil percobaan menggunakan *threshold*= 0.1, *step_size*= 0.1 dan 0.05 adalah sama. Hasil uji coba menggunakan *threshold*= 0.1, *step_size*= 0.1 dan 0.05 dapat dilihat pada Tabel 5.2. Hasil uji coba menggunakan *threshold*= 0.1, *step_size*= 0.025 dapat dilihat pada Tabel 5.3. Kolom yang diberi warna hijau adalah kolom yang memuat nilai indeks Dunn lebih besar dibanding kolom lainnya.

Tabel 5.2. Tabel Hasil Uji Coba 1 (threshold = 0.1, step_size =0.1 dan 0.05)

Percobaan ke-	Indeks Dunn *FMST*	Indeks Dunn *MST*	Indeks Dunn Kmeans	Jumlah Klaster yang dihasilkan *FMST*	Jumlah Klaster yang dihasilkan *MST*
1	0.184	0.184	0.033	2	2
2	0.183	0.183	0.045	2	2
3	0.232	0.232	0.014	2	2
4	0.036	0.141	0.01	4	3
5	0.105	0.105	0.058	4	4
6	0.474	0.474	0.474	5	5
7	0.085	0.085	0.022	5	5
8	0.085	0.085	0.034	5	5
9	0.085	0.085	0.029	5	5
10	0.257	0.257	0.257	2	2
11	0.134	0.134	0.006	3	3
12	0.123	0.123	0.012	2	2
13	0.175	0.175	0.022	3	3
14	0.203	0.203	0.203	2	2
15	0.156	0.156	0.022	4	4

Tabel 5.3. Tabel Hasil Uji Coba 2 (threshold = 0.1, step_size =0.025)

Percobaan ke-	Indeks Dunn *FMST*	Indeks Dunn *MST*	Indeks Dunn Kmeans	Jumlah Klaster yang dihasilkan *FMST*	Jumlah Klaster yang dihasilkan *MST*
1	0.184	0.184	0.033	2	2
2	0.183	0.183	0.045	2	2
3	0.053	0.053	0.056	6	6
4	0.025	0.043	0.016	3	4
5	0.105	0.105	0.058	4	4
6	0.474	0.474	0.474	5	5
7	0.085	0.085	0.022	5	5
8	0.085	0.085	0.034	5	5
9	0.085	0.085	0.029	5	5
10	0.257	0.257	0.257	2	2
11	0.134	0.134	0.006	3	3
12	0.123	0.123	0.012	2	2
13	0.175	0.175	0.022	3	3
14	0.203	0.203	0.203	2	2
15	0.156	0.156	0.022	4	4

5.3.2 Skenario Uji Coba dengan Beberapa *Dataset*

Uji coba dilakukan terhadap dua *dataset* yang umum digunakan yaitu, data *iris* dan data *wine*. Uji coba dilakukan sebanyak 20 kali pada tiap data. Waktu dan *weight error* yang didapat pada uji coba tercatat pada Tabel 5.4. Setelah itu, dilakukan klastering berdasarkan *MST* yang didapat dari metode *FMST* dan algoritma konvensional *MST*, dan juga menggunakan algoritma

klastering K-Means. Parameter jumlah klaster yang dipakai algoritma K-Means adalah jumlah klaster yang dihasilkan oleh klastering berdasarkan algoritma *MST* konvensional. Hasil klastering yang diperoleh dievaluasi menggunakan Indeks Dunn. Skenario Uji Coba dilakukan dengan *threshold* bernilai 0.1 dan *step_size* bernilai 0.1, 0.05, dan 0.025. Setelah dilakukan uji coba, ternyata hasil percobaan menggunakan *threshold*= 0.1, *step_size*= 0.1, 0.05, dan 0.025 adalah sama. Hasil uji coba tersebut dapat dilihat pada Tabel 5.5. Pada Tabel 5.5, kolom yang diberi warna hijau adalah kolom yang memuat nilai indeks Dunn lebih besar dibanding kolom lainnya. Pada Tabel 5.5 baik data *iris* maupun *wine*, klastering yang dilakukan menghasilkan 2 klaster, sesuai dengan nilai Indeks Dunn terbesar yang didapatkan. Sedangkan, menurut *ground truth* nya, kedua *dataset* tersebut terdiri dari 3 klaster. Maka dari itu dilakukan skenario coba dilakukan evaluasi klastering menggunakan klastering *FMST*, *MST* dan Kmeans dengan "memaksakan" data dipartisi menjadi 3 klaster. Nilai evaluasi tersebut dapat dilihat pada Tabel 5.6.

Selain Indeks Dunn, hasil klastering juga dinilai melalui *ground truth.* Label klaster yang dihasilkan oleh klastering berdasarkan *FMST*, klastering berdasarkan *MST*, K-Means dicocokkan dengan label klaster *ground truth.* Hasil perbandingan tersebut dapat di lihat pada Tabel 5.7, Tabel 5.8, Tabel 5.9, dan Tabel 5.10. Pada tabel – tabel tersebut, data dikelompokkan menurut hasil klastering prediksi dan hasil klastering sebenarnya. Sebagai contoh, pada Tabel 5.8, dengan menggunakan klastering berdasarkan *FMST*, *dataset Iris* dibagi menjadi 3 klaster. Klaster prediksi 1 memuat 50 data yang semuanya berasal dari klaster 1 yang sebenarnya. Klaster prediksi 2 memuat 98 data, terdiri dari 50 data berasal dari klaster 2 sesungguhnya dan 48 data yang berasal

dari klaster 3 sesungguhnya. Klaster prediksi 3, terdiri dari 2 data yang seluruhnya berasal dari klaster 3 sesungguhnya.

Tabel 5.4. Tabel Perbandingan Waktu dan weight error

	Waktu yang diperlukan *FMST* (hasil terbaik)	**Waktu rata-rata yang diperlukan *FMST***	**Waktu yang diperlukan *MST***	***Weight error* minimum**	***Weight error* rata-rata**
Iris	2.088	1.787	4.972	0	0.426
Wine	2.073	2.166	7.944	0	0.074

Tabel 5.5. Tabel Hasil Uji Coba 1 terhadap Data Real

	Indeks Dunn *FMST*	**Indeks Dunn *MST***	**Indeks Dunn K-Means**	**Jumlah Klaster yang dihasil-kan *FMST***	**Jumlah Klaster yang dihasil-kan *MST***
Iris	0.339	0.339	0.077	2	2
Wine	0.105	0.105	0.023	2	2

Tabel 5.6. Tabel Hasil Uji Coba 2 terhadap Data Real

	Indeks Dunn *FMST*	Indeks Dunn *MST*	Indeks Dunn K-Means
Iris	0.169	0.169	0.099
Wine	0.068	0.068	0.016

Tabel 5.7. Tabel Hasil klastering pada Uji Coba 1 (Dataset Iris)

Klaster sebenarnya	*Ground Truth*	Klaster prediksi					
		FMST		*MST*		K-Means	
		1	2	1	2	1	2
1	50	50	0	50	0	50	0
2	50	0	50	0	50	3	47
3	50	0	50	0	50	0	50

Tabel 5.8. Tabel Hasil klastering pada Uji Coba 2 (Dataset Iris)

Klaster Sebenar-nya	*Ground Truth*	Klaster prediksi								
		FMST			*MST*			K-Means		
		1	2	3	1	2	3	1	2	3
1	50	50	0	0	50	0	0	50	0	0
2	50	0	50	0	0	50	0	0	48	2
3	50	0	48	2	0	48	2	0	14	36

Tabel 5.9. Tabel Hasil klastering pada Uji Coba 1 (Dataset Wine)

Klaster Sebenar nya	*Ground Truth*	Klaster prediksi					
		FMST		*MST*		Kmeans	
		1	2	1	2	1	2
1	59	1	58	1	58	50	9
2	71	0	71	0	71	4	67
3	48	0	48	0	48	1	47

Tabel 5.10. Tabel Hasil klastering pada Uji Coba 2 (Dataset Wine)

Klaster sebenarnya	*Ground Truth*	Klaster prediksi								
		FMST			*MST*			K-Means		
		1	2	3	1	2	3	1	2	3
1	59	5	53	1	5	53	1	46	0	13
2	71	0	71	0	0	71	0	1	50	20
3	48	0	48	0	0	48	0	0	19	29

5.4 Analisis Hasil Uji Coba

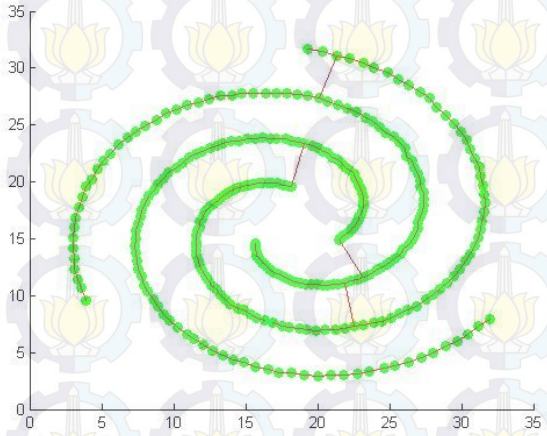
Analisa dilakukan tersendiri pada data sintesis dan data real. Analisa dilakukan berdasarkan hasil dari tiap skenario uji coba dengan evaluasi tiap pengujian.

5.4.1 Analisa Hasil Uji Coba Data Sintesis

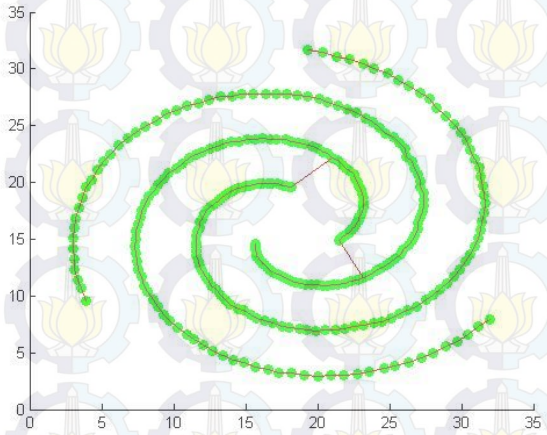
Berdasarkan Gambar 5.1 dan Gambar 5.2, diketahui bahwa metode *FMST* dapat menyelesaikan permasalahan *MST* dengan waktu yang lebih sedikit. Untuk masalah akurasi, dapat dilihat pada Gambar 5.3, bahwa 13 dari 15 percobaan memiliki *weight error* dibawah 1 persen. Hasil percobaan ke 4 dan ke 11 pada data sintesis yang memiliki *weight error* rata-rata diatas 1 persen dikarenakan kelompok-kelompok pada data tersebut tidak terpisah jauh. Visualisasi *FMST* pada percobaan ke 4 dapat dilihat pada Gambar 5.7, serta *MST* sebenarnya pada percobaan ke 4 dapat dilihat pada Gambar 5.8. Terlihat perbedaan mencolok pada 2 gambar tersebut.

Percobaan ke 3, 7, dan 11 memiliki *weight error* rata-rata hampir mendekati 0 persen. Hal ini dikarenakan kelompok-kelompok pada data tersebut terpisah secara baik. Visualisasi *FMST* pada percobaan ke 3 dapat dilihat pada Gambar 5.9, serta

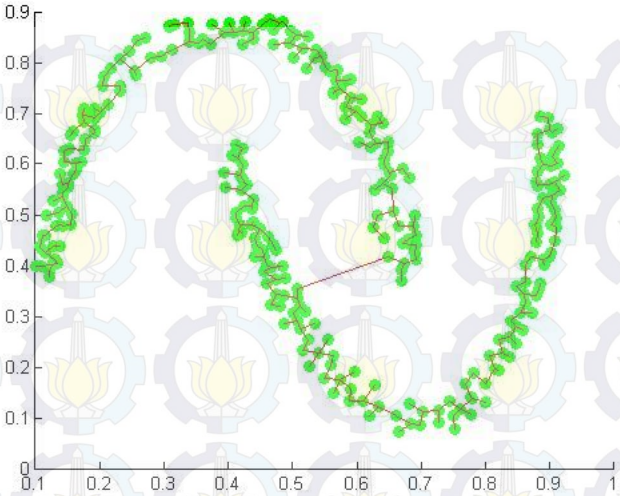
MST sebenarnya pada percobaan ke 3 dapat dilihat pada Gambar 5.10. Tidak ada perbedaan diantara 2 gambar tersebut.



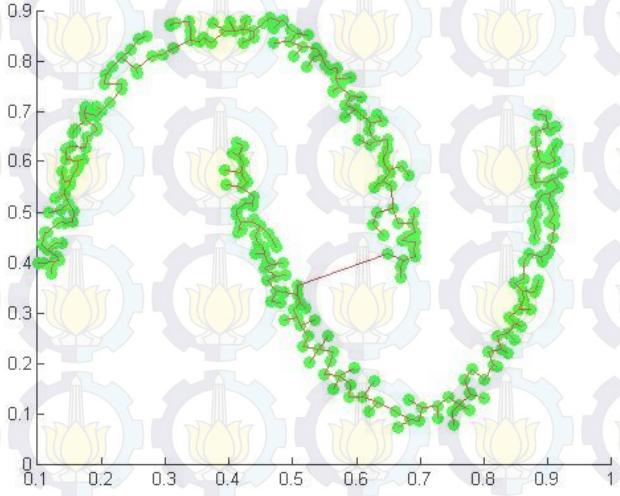
Gambar 5.7. Hasil FMST terbaik pada percobaan ke 4



Gambar 5.8. Hasil MST pada percobaan ke 4

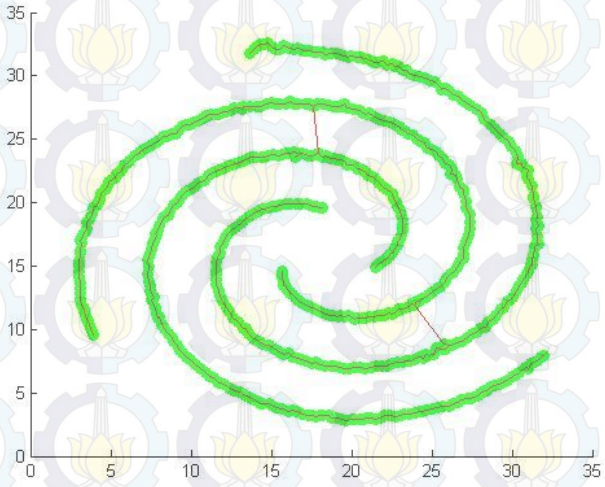


Gambar 5.9. Hasil FMST terbaik pada percobaan ke 3

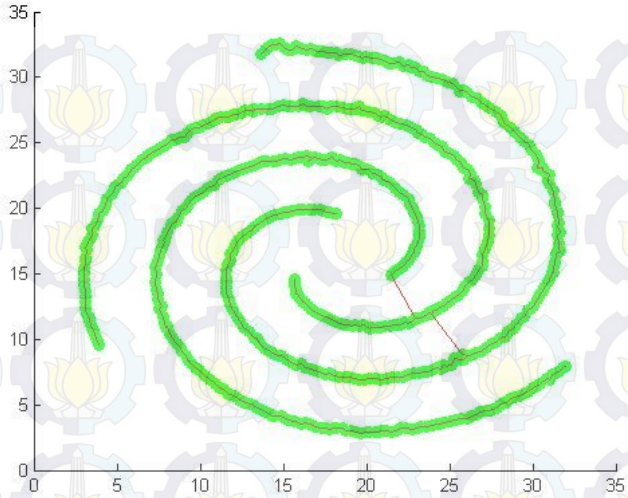


Gambar 5.10. Hasil MST pada percobaan ke 3

Kerapatan antar data juga menjadi faktor penting agar metode *FMST* dapat menyelesaikan permasalahan *MST* dengan baik. Hal ini dapat dilihat pada percobaan ke 4 dan ke 11 pada data sintesis, data pada percobaan tersebut memiliki bentuk yang sama, hanya jumlah data yang berbeda. Percobaan ke 4 yang memiliki 312 data menghasilkan *weight error* lebih tinggi dibanding dengan percobaan ke 11 yang memiliki 612 data. Semakin banyak data, maka semakin besar kelompok yang dihasilkan pada tahap *Divide and Conquer*. Semakin besar kelompok pada tahap *Divide and Conquer*, maka semakin besar pula kemungkinan *subset* yang dihubungkan dengan *subset* yang benar pada tahap *Combine Subset Algorithm*. Visualisasi *FMST* pada percobaan ke 11 dapat dilihat pada Gambar 5.11, serta *MST* sebenarnya pada percobaan ke 11 dapat dilihat pada Gambar 5.12.



Gambar 5.11. Hasil FMST terbaik pada percobaan ke 11

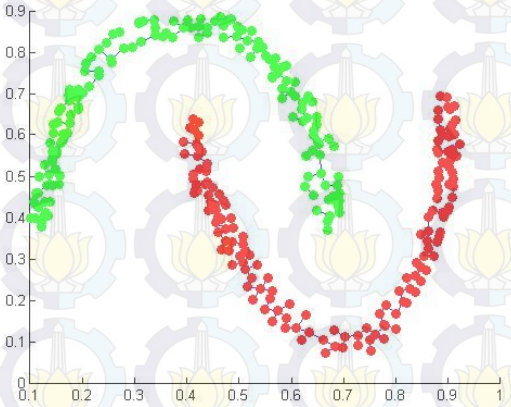


Gambar 5.12. Hasil MST pada percobaan ke 11

Dilihat dari Tabel 5.1, Tabel 5.2, dan Tabel 5.3, skenario uji coba dengan menggunakan *threshold= 0.1* dan *step_size= 0.05* atau *0.1* menghasilkan hasil klastering lebih baik dibandingkan skenario uji coba menggunakan *threshold= 0.1* dan *step_size= 0.025*. Hal ini ditunjukkan dari 15 percobaan menggunakan *step_size= 0.05* atau *0.1*, 13 percobaan diantaranya menunjukkan jumlah klaster yang dihasilkan klastering berdasarkan *MST* sama dengan jumlah klaster sebenarnya. Percobaan pada uji coba 1 yang gagal mengidentifikasi jumlah klaster adalah percobaan 2, dan 14. Sedangkan, jika menggunakan *step_size= 0.025*, hanya 11 dari 15 percobaan yang menunjukkan jumlah klaster yang dihasilkan klastering berdasarkan *MST* sama dengan jumlah klaster sebenarnya. Percobaan pada uji coba 2 yang gagal mengidentifikasi jumlah klaster adalah percobaan 2, 3, 4, dan 14.

Ditinjau dari Nilai Indeks Dunn, Tabel 5.2, dan Tabel 5.3 didominasi dengan lebih bagusnya hasil klastering berdasarkan *MST* dibanding Kmeans. Satu-satunya percobaan yang

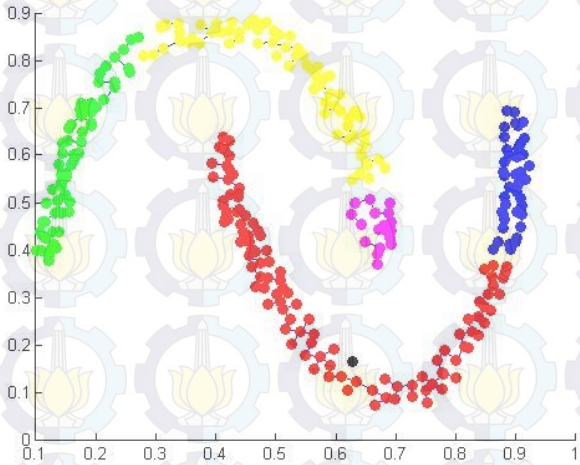
membedakan adalah pada percobaan ke 3, dimana pada uji coba ke 2 untuk data sintesis menunjukkan hasil klastering K-Means menghasilkan indeks Dunn lebih besar. Visualisasi tersebut dapat dilihat pada Gambar 5.15.Visualisasi *FMST* pada uji coba 1 untuk percobaan ke 3 pada data sintesis dapat dilihat pada Gambar 5.13, serta *MST* sebenarnya pada uji coba 2 untuk percobaan ke 3 dapat dilihat pada Gambar 5.14.



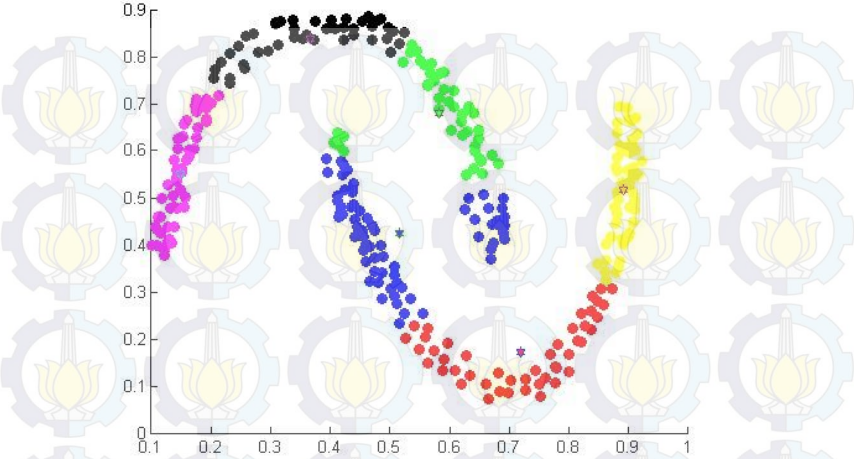
Gambar 5.13. Hasil klastering menggunakan FMST pada uji coba 1 untuk percobaan ke 3

Dua percobaan yang sama-sama gagal diidentifikasi Tabel Hasil Uji Coba 1 (*threshold* = 0.1, *step_size* =0.1 dan 0.05), dan Tabel Hasil Uji Coba 2 (*threshold* = 0.1, *step_size* =0.025) dengan benar dalam hal jumlah klaster nya adalah percobaan ke 2 dan ke 14. Percobaan tersebut, jumlah klasternya gagal diidentifkasi dengan benar menggunakan klastering berdasarkan *FMST*, maupun berdasarkan *MST*. Jumlah klaster yang sebenarnya pada *dataset* adalah 3, namun hanya diidentifikasi terdiri dari 2 klaster. Hal ini mungkin diakibatkan klaster – klaster yang ada tidak terletak begitu jauh. Sebagai argumen, percobaan ke 13 pada data

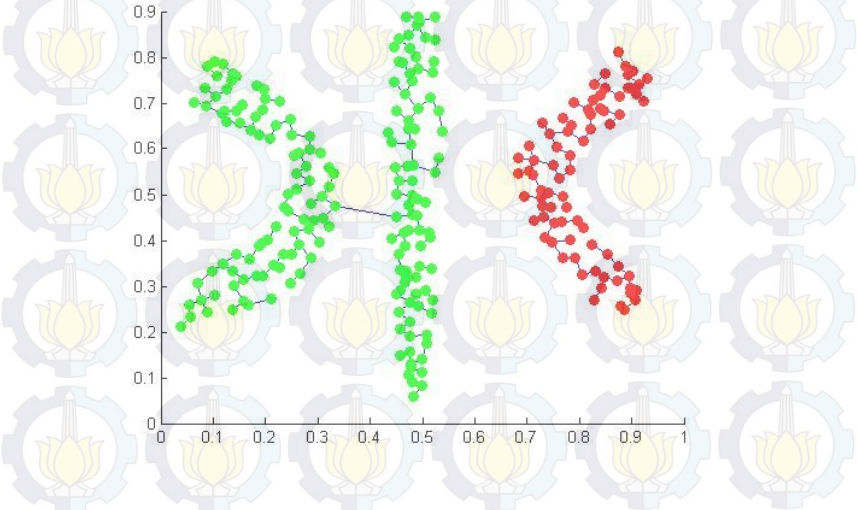
sintesis memiliki bentuk *dataset* yang sama dengan percobaan ke 14 pada data sintesis. Jumlah data pada *dataset* pun sama-sama 700. Namun *dataset* percobaan ke 13, klaster-klasternya terletak lebih berjauhan dibanding percobaan ke 14. Sehingga percobaan ke 13 mengembalikan jumlah klaster yang tepat dengan jumlah klaster sebenarnya, yaitu 3. Visualisasi hasil klastering berdasarkan *FMST*/ *MST* percobaan ke 2 dapat dilihat pada Gambar 5.16. Visualisasi kasil klastering berdasarkan *FMST*/ *MST* percobaan ke 14 dapat dilihat pada Gambar 5.17. Visualisasi hasil klastering berdasarkan *FMST* / *MST* percobaan ke 13 dapat dilihat pada Gambar 5.18.



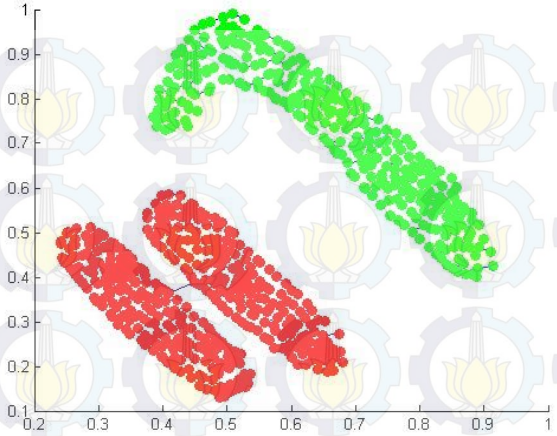
Gambar 5.14. Hasil klastering menggunakan FMST pada uji coba 2 untuk percobaan ke 3



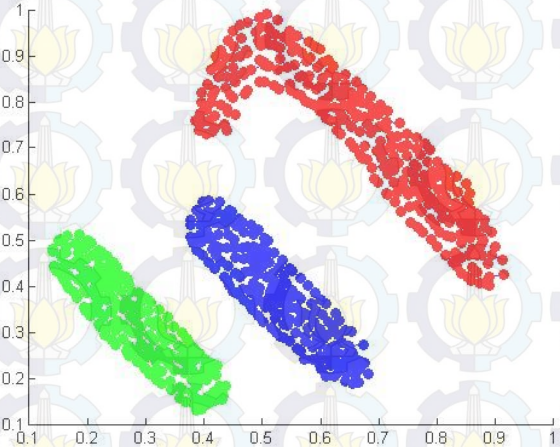
Gambar 5.15. Hasil klastering menggunakan Kmeans pada uji coba 2 untuk percobaan ke 3



Gambar 5.16. Hasil klastering menggunakan FMST/MST pada percobaan ke 2

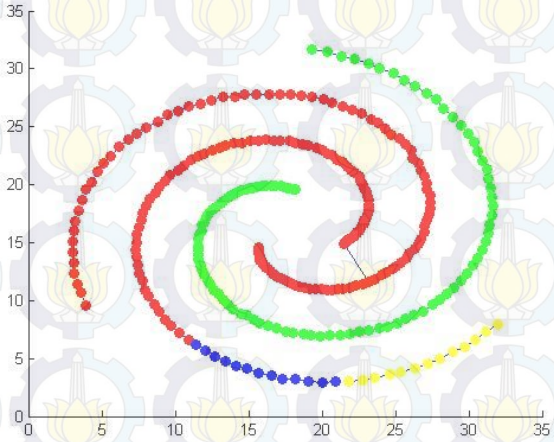


Gambar 5.17. Hasil klastering menggunakan FMST/MST pada percobaan ke 14



Gambar 5.18. Hasil klastering menggunakan FMST/MST pada percobaan ke 13

Klastering berdasarkan *FMST* tidak kalah baik dibanding klastering berdasarkan *MST*. Dapat dilihat pada Tabel 5.2, dan Tabel 5.3, klastering berdasarkan *FMST* kalah dibanding klastering berdasarkan *MST* di percobaan ke 4. Visualisasi hasil klastering berdasarkan *FMST*/ *MST* percobaan ke 4 dapat dilihat pada Gambar 5.19. Hal ini ditengarai akibat *weight error* percobaan ke 4 bernilai relatif tinggi.



Gambar 5.19. Hasil klastering menggunakan FMST pada percobaan ke 4 uji coba 1

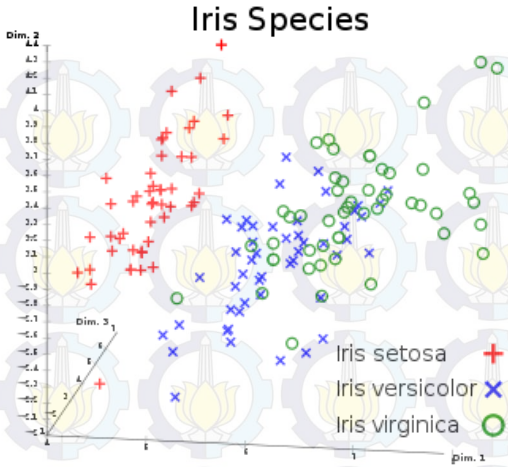
5.4.2 Analisa Hasil Uji Coba Data Real

Menurut Tabel 5.5, hasil indeks Dunn klastering menggunakan *FMST*, maupun *MST* pada *dataset iris*, dan *wine* bernilai lebih besar dibanding hasil klastering K-Means. Hal ini dapat dilihat pula pada Tabel 5.6, dimana uji coba dilakukan menggunakan jumlah klaster yang sama jumlah klaster sesungguhnya, yaitu 3 klaster. Pada tabel tersebut, hasil indeks Dunn klastering menggunakan *FMST*, maupun *MST* pada *dataset iris* dan *wine* tetap bernilai besar dari hasil K-Means. Kedua uji

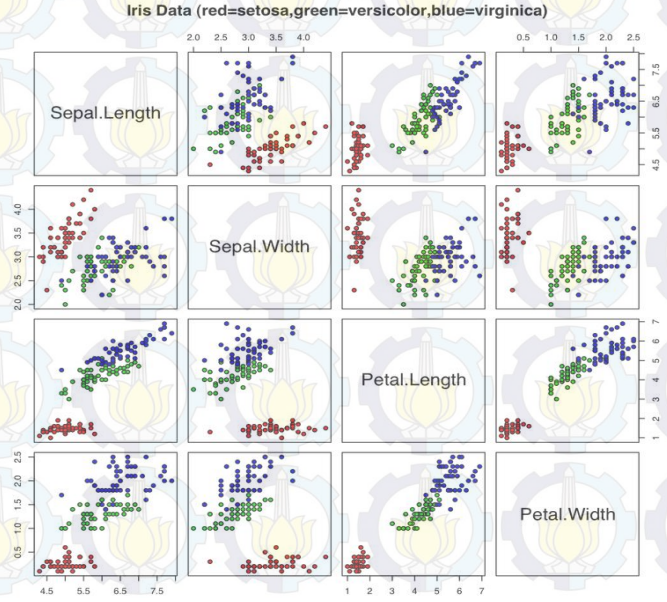
coba tersebut menandakan bahwa menggunakan klastering berdasarkan *MST* lebih baik dibanding menggunakan K-Means.

Akan, tetapi, K-Means terbilang lebih baik dalam hal mengelompokkan data ke klaster yang sesuai dengan klaster sebenarnya. Hal ini dapat dilihat pada Tabel 5.8, dan Tabel 5.10. Pada Tabel 5.8, klastering berdasarkan *FMST*, maupun *MST* hanya dapat mengelompokkan 102 data ke klaster yang tepat. Sedangkan K-Means mampu mengelompokkan 134 data ke klaster yang tepat. Pada Tabel 5.10, klastering berdasarkan *FMST*, maupun *MST* hanya dapat mengelompokkan 76 data ke klaster yang tepat. Sedangkan K-Means mampu mengelompokkan 125 data ke klaster yang tepat. Hal ini mungkin disebabkan dataset Iris dan Wine memiliki *overlapping* yang tinggi. *FMST/ MST* berhasil mengelompokkan *dataset* Iris dengan baik ketika *dataset* dibagi menjadi 2 klaster. Hal ini dapat dilihat pada Tabel 5.7, bahwa hasil klastering menggunakan *FMST/ MST* lebih baik daripada K-Means. Sedangkan dengan *dataset* Wine pada Tabel 5.8, ketika diklastering menggunakan *FMST/ MST* tidak mengelompokkan data lebih baik dibanding menggunakan K-Means.

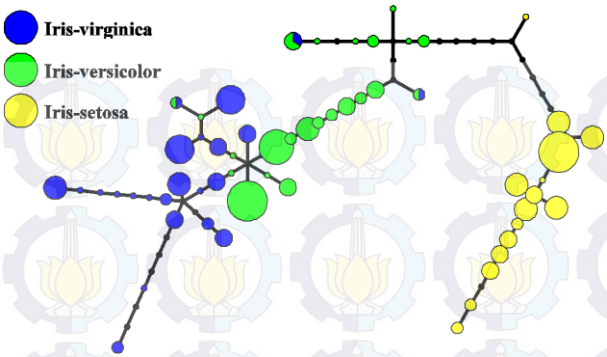
Dapat dilihat pada Gambar 5.20, Gambar 5.21, dan Gambar 5.22 [22], *dataset* Iris di-*plot* ke beberapa macam representasi. Kelas *iris-setosa* memiliki tingkat kemiripan yang sangat jauh dengan dua kelas lainnya. Sedangkan kelas *iris-versicolor*, dan kelas *iris-virginica* saling tumpah tindih (*overlap*). Sehingga ketika dataset Iris dilakukan klastering menggunakan *FMST/ MST*, meskipun terbentuk klaster yang memiliki *separation* baik (ditandai dengan nilai indeks Dunn yang relatif tinggi dibanding K-Means), dataset Iris tidak terkelompokkan secara tepat.



Gambar 5.20. Ilustrasi dataset Iris



Gambar 5.21. Ilustrasi dataset Iris dibandingkan tiap atribut



Gambar 5.22. Ilustrasi "metromap" dataset Iris

[Halaman ini sengaja dikosongkan]

BAB VI
KESIMPULAN DAN SARAN

Bab ini membahas mengenai kesimpulan yang dapat diambil dari hasil uji coba yang telah dilakukan sebagai jawaban dari rumusan masalah yang dikemukakan. Selain kesimpulan, juga terdapat saran yang ditujukan untuk pengembangan penelitian lebih lanjut.

6.1 Kesimpulan

Dari hasil uji coba yang telah dilakukan terhadap pembuatan model, dapat diambil kesimpulan sebagai berikut:

1. Metode *Fast Minimum Spanning Tree* (*FMST*) dapat digunakan untuk menyelesaikan *MST* lebih cepat dibanding algoritma *MST* konvensional dengan *weight error* rata-rata 0.988%.
2. Klastering berdasarkan *FMST* mendapatkan hasil terbaik ketika menggunakan *threshold*= 0.1 dan *step_size*= 0.1.
3. Klastering menggunakan *FMST* mampu mengelompokkan dataset yang berbentuk n*on-convex.* Akan lebih optimal jika data memiliki jarak antar klaster dan kerapatan data cukup tinggi. Akan tetapi, *FMST* kurang optimal digunakan mengelompokkan data yang memiliki *overlap* tinggi.

6.2 Saran

Saran yang diberikan untuk pengembangan aplikasi ini adalah:

1. Untuk meningkatkan waktu pemrosesan, dapat digunakan bahasa pemrograman yang lain.
2. Perlu penelitian lebih lanjut mengenai tahap *Combine Subset Algorithm* untuk meningkatkan akurasi *FMST*.
3. Untuk meningkatkan akurasi hasil klastering, dapat dilakukan metode klastering berdasarkan *MST* yang lain.

[Halaman ini sengaja dikosongkan]

DAFTAR PUSTAKA

[1] The MathWorks, Inc, [Online]. Available: http://www.mathworks.com/discovery/pattern-recognition.html.

[2] Y. Sitohang. [Online]. Available: http://yukisuhendrasitohang.blogspot.com/2010/10/clustering-adalah-proses-mengelompokkan.html.

[3] P. Adhitama, "Kovloq [Never stop to (l)earn and share]," [Online]. Available: http://www.kovloq.com/2013/04/24/image-segmentation-using-k-means/.

[4] E. Barse, H. Kvarnström and E. Jonsson, *Synthesizing test data for fraud detection systems,* pp. 1-11, 2003.

[5] V. Estivill-Castro, "Why so many clustering algorithms," *Why so many clustering algorithms,* vol. 4, no. 1, pp. 65-75, 2002.

[6] Z. Caiming, M. Mikko, M. Dioqian and F. Pasi, "A Fast Minimum Spanning Tree Algorithm based on K-Means," 2014.

[7] P. K.Jana and A. Naik, "An Efficient Minimum Spanning Tree based Clustering Algorithm," 2009.

[8] D. Rachmawati, "Dee's Personal Pages," [Online]. Available: https://dee83.wordpress.com/2008/08/16/konsep-dasar-graph/.

[9] Saluky, "Saluky @etunas Best Solution," [Online]. Available: http://saluky.blogspot.com/2012/04/teori-graf.html.

[10] D. Rahardi, "SCIENCE FOR HUMANITY : Minimal Spanning Tree," [Online]. Available: http://dickyrahardi.blogspot.com/2008/05/minimal-spanning-tree.html.

[11] M. Fellows and N. Casey, "MegaMath," [Online]. Available: http://www.c3.lanl.gov/mega-math/gloss/graph/grpath.html.

[12] I. Firmansyah, "Algoritma Prims," [Online]. Available: https://sikomku.wordpress.com/2011/02/23/algoritma-prims/.

[13] Wolfram Research, Inc. , "Convex Set --from Mathworld Wolfram," [Online]. Available: http://mathworld.wolfram.com/ConvexSet.html.

[14] Encyclopædia Britannica, Inc, [Online]. Available: http://www.britannica.com/EBchecked/topic/135838/convex-set/images-videos/1737/convex-set-euclidean-geometry.

[15] GeeksforGeeks, "Greedy Algorithms | Set 5 (Prim's Minimum Spanning Tree (MST))," [Online]. Available: http://www.geeksforgeeks.org/greedy-algorithms-set-5-prims-minimum-spanning-tree-mst-2/.

[16] F. Dita, "Tahap-tahap K-Means Clustering," [Online]. Available: https://fadlikadn.wordpress.com/2013/06/14/tahap-tahap-k-means-clustering/.

[17] S. Towers, "Polymatheia," [Online]. Available: http://sherrytowers.com/2013/10/24/k-means-clustering/.

[18] E. Irwansyah, in *Advanced Clustering: Teori dan Aplikasi*, Sleman, DEEPUBLISH, 2015, p. 17.

[19] "Análisis y Manejo de Información," [Online]. Available: http://analisisymanejodeinformacion.blogspot.com/2011_12_08_archive.html.

[20] Ismail, "Berbagi Cerita & Kebahagiaan," [Online]. Available: https://basayevsmile.files.wordpress.com/2008/09/bab-6-visualisasi-data.pdf.

[21] National Science Foundation, [Online]. Available: http://archive.ics.uci.edu/ml/datasets.html.

[22] Wikipedia, "Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Iris_flower_data_set.

[Halaman ini sengaja dikosongkan]

BAB VI
KESIMPULAN DAN SARAN

Bab ini membahas mengenai kesimpulan yang dapat diambil dari hasil uji coba yang telah dilakukan sebagai jawaban dari rumusan masalah yang dikemukakan. Selain kesimpulan, juga terdapat saran yang ditujukan untuk pengembangan penelitian lebih lanjut.

6.1 Kesimpulan

Dari hasil uji coba yang telah dilakukan terhadap pembuatan model, dapat diambil kesimpulan sebagai berikut:

1. Metode *Fast Minimum Spanning Tree* (*FMST*) dapat digunakan untuk menyelesaikan *MST* lebih cepat dibanding algoritma *MST* konvensional dengan *weight error* rata-rata 0.988%.
2. Klastering berdasarkan *FMST* mendapatkan hasil terbaik ketika menggunakan *threshold*= 0.1 dan *step_size*= 0.1.
3. Klastering menggunakan *FMST* mampu mengelompokkan dataset yang berbentuk n*on-convex.* Akan lebih optimal jika data memiliki jarak antar klaster dan kerapatan data cukup tinggi. Akan tetapi, *FMST* kurang optimal digunakan mengelompokkan data yang memiliki *overlap* tinggi.

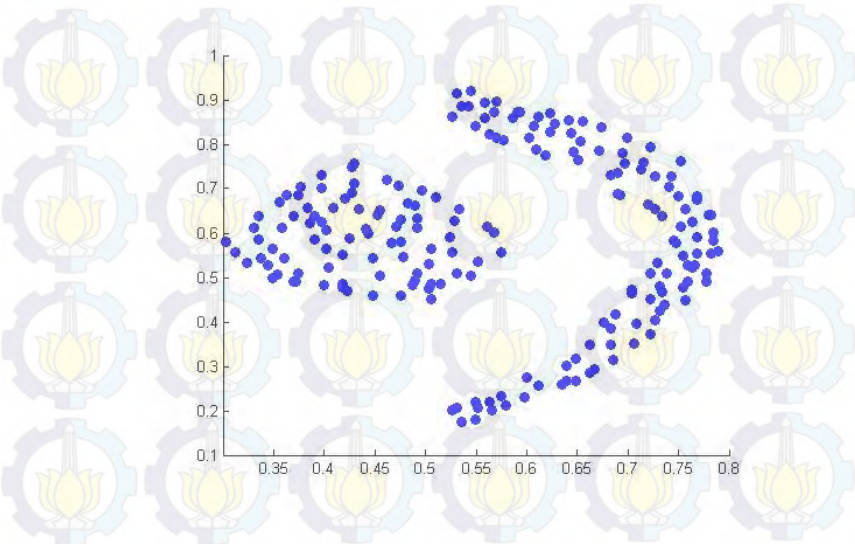
6.2 Saran

Saran yang diberikan untuk pengembangan aplikasi ini adalah:

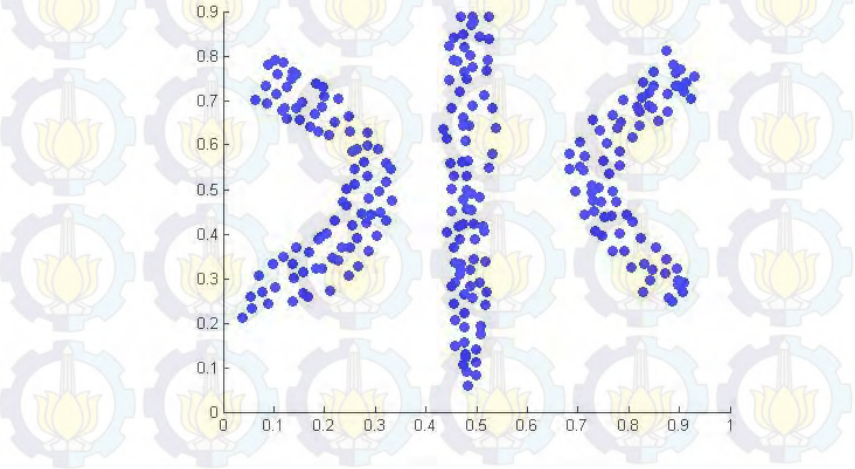
1. Untuk meningkatkan waktu pemrosesan, dapat digunakan bahasa pemrograman yang lain.
2. Perlu penelitian lebih lanjut mengenai tahap *Combine Subset Algorithm* untuk meningkatkan akurasi *FMST*.
3. Untuk meningkatkan akurasi hasil klastering, dapat dilakukan metode klastering berdasarkan *MST* yang lain.

[Halaman ini sengaja dikosongkan]

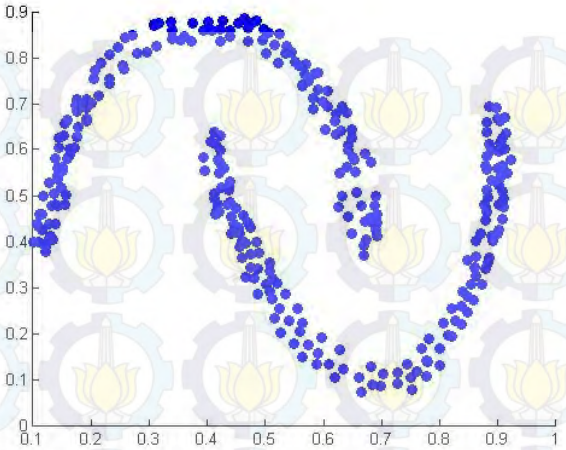
LAMPIRAN



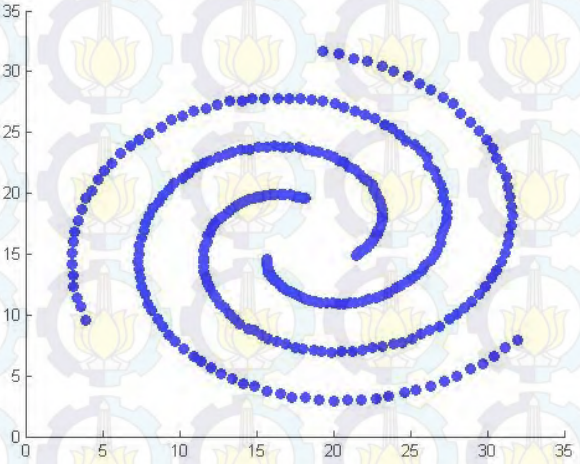
Gambar A.1. Visualisasi Dataset T1



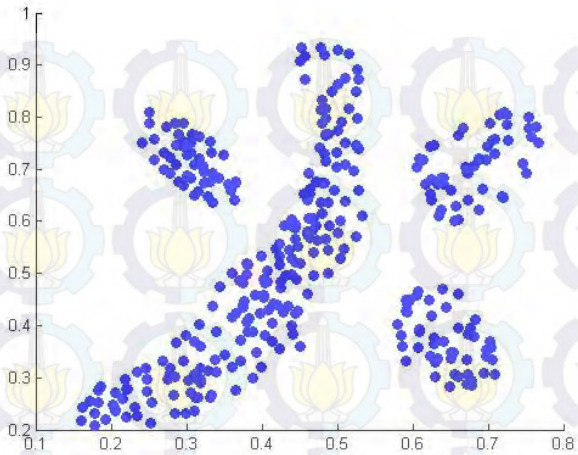
Gambar A.2. Visualisasi Dataset T2



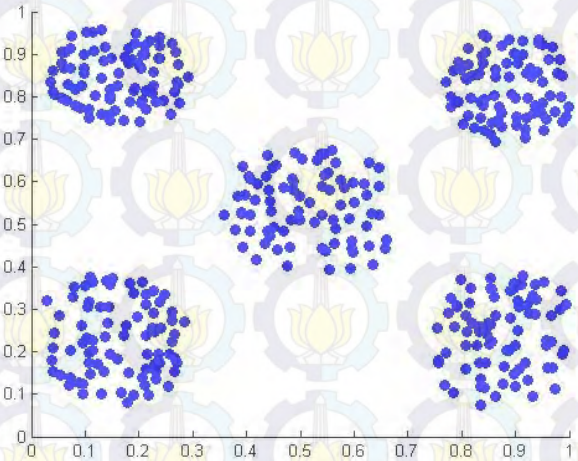
Gambar A.3. Visualisasi Dataset Curve



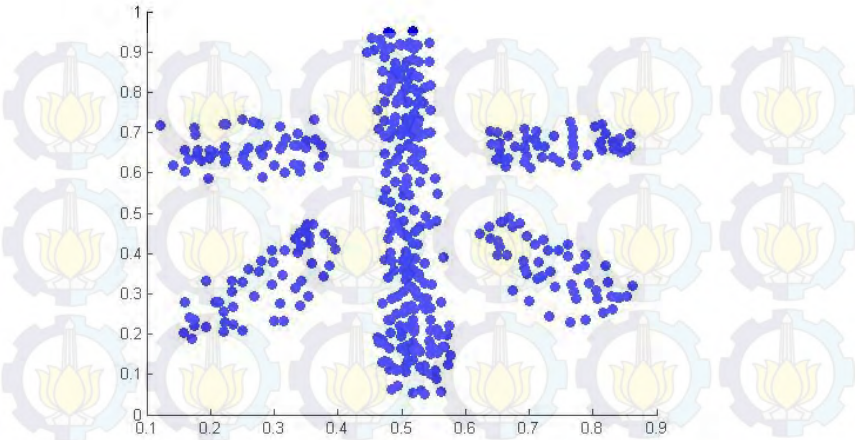
Gambar A.4. Visualisasi Dataset Spiral1



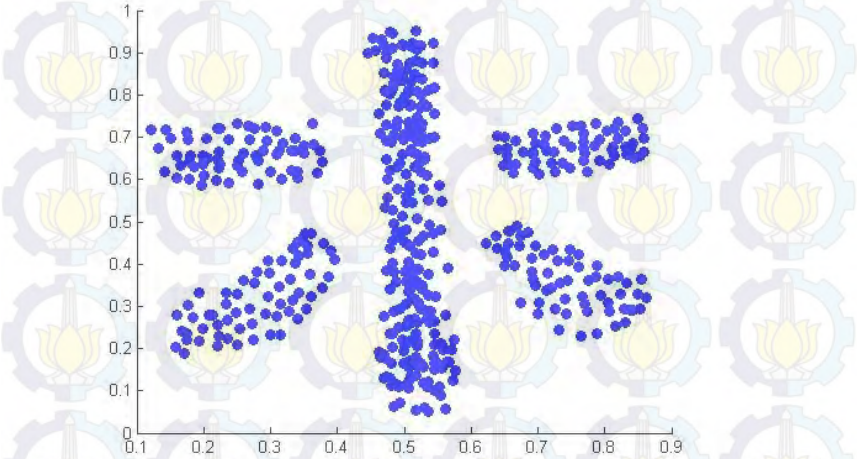
Gambar A.5. Visualisasi Dataset T3



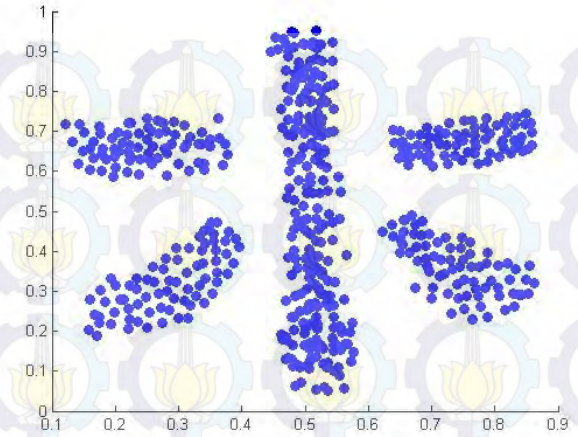
Gambar A.6. Visualisasi Dataset T4



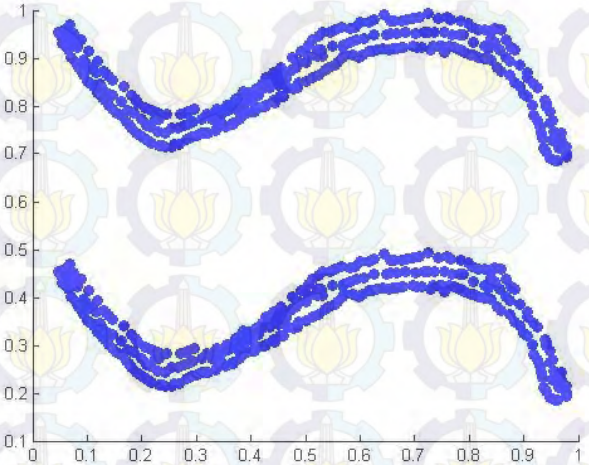
Gambar A.7. Visualisasi Dataset T5



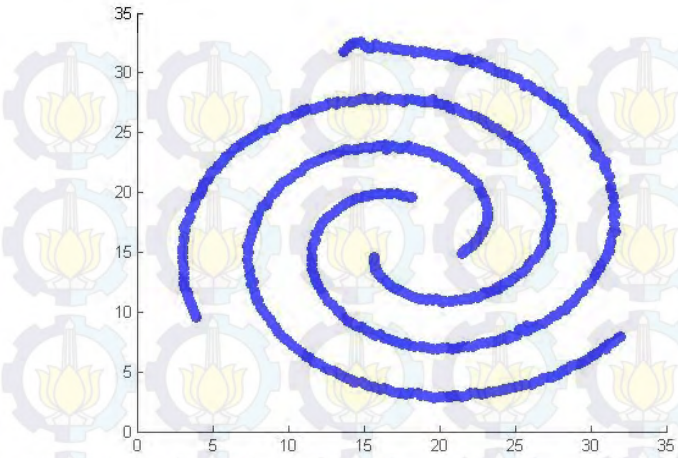
Gambar A.8. Visualisasi Dataset T6



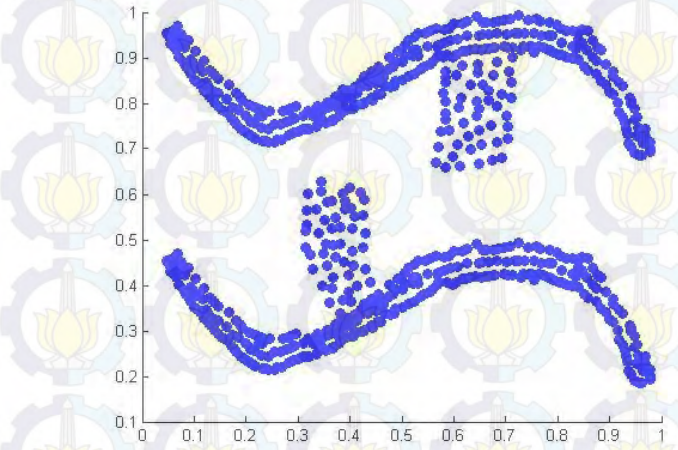
Gambar A.9. Visualisasi Dataset T7



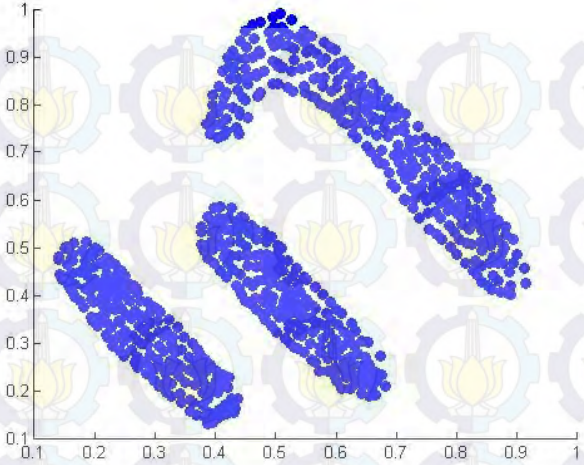
Gambar A.10. Visualisasi Dataset T8



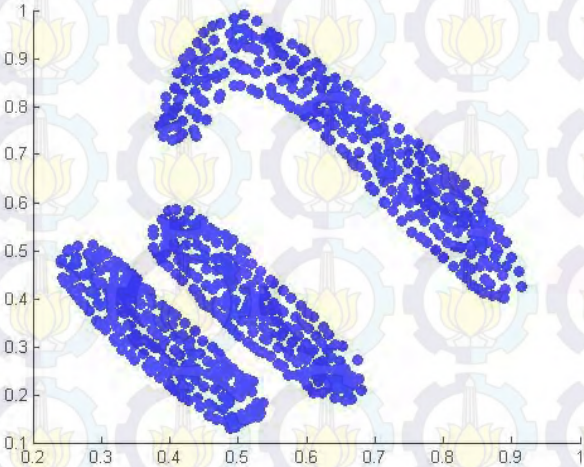
Gambar A.11. Visualisasi Dataset Spiral2



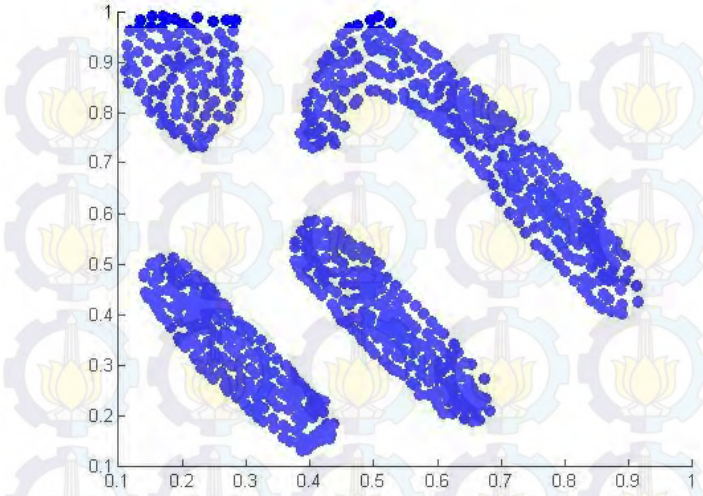
Gambar A.12. Visualisasi Dataset T9



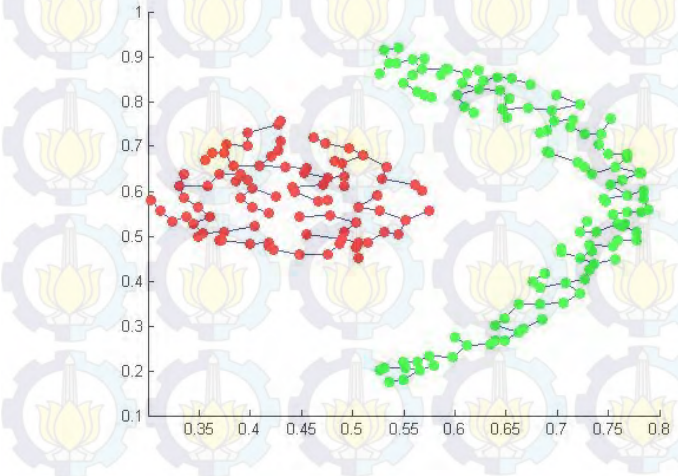
Gambar A.13. Visualisasi Dataset T10



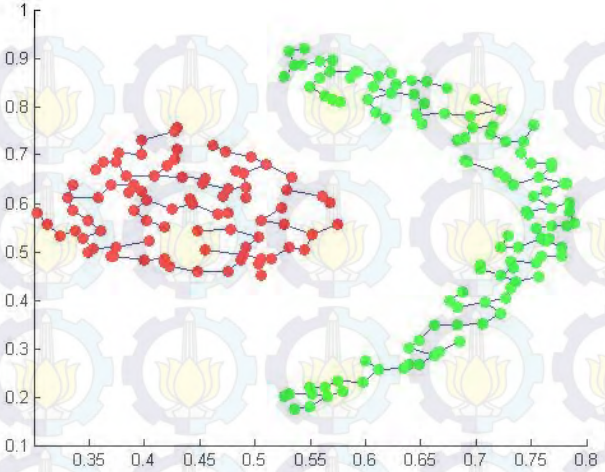
Gambar A.14. Visualisasi Dataset T11



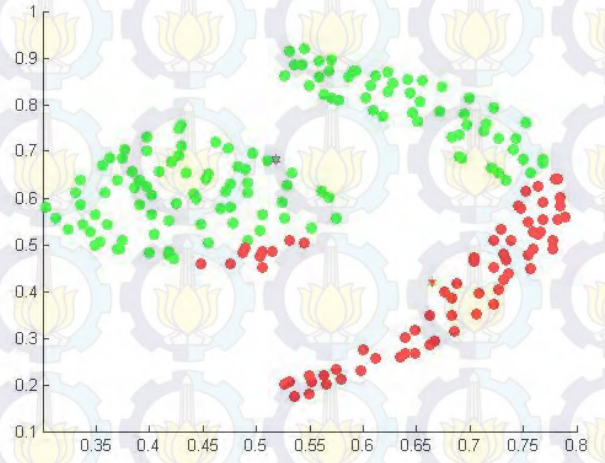
Gambar A.15. Visualisasi Dataset T12



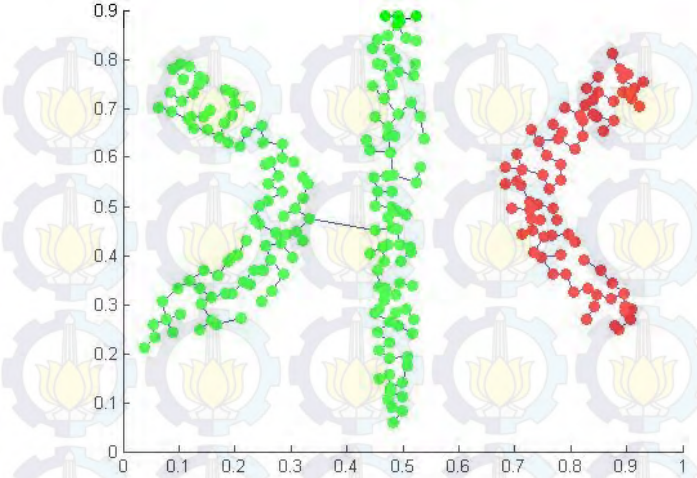
Gambar A.16. Hasil Uji Coba 1 menggunakan FMST pada percobaan ke-1



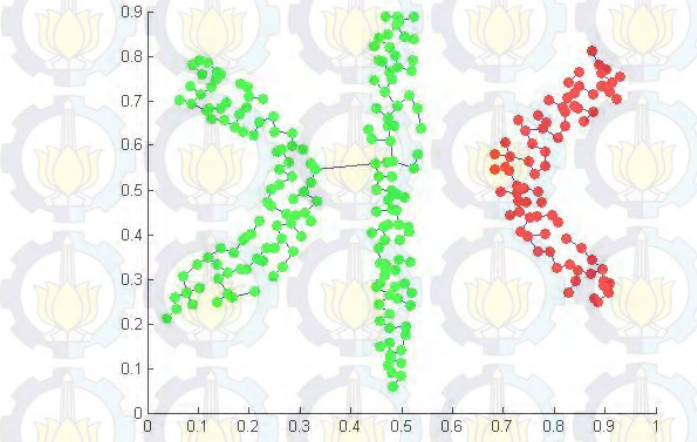
Gambar A.17. Hasil Uji Coba 1 menggunakan MST pada percobaan ke-1



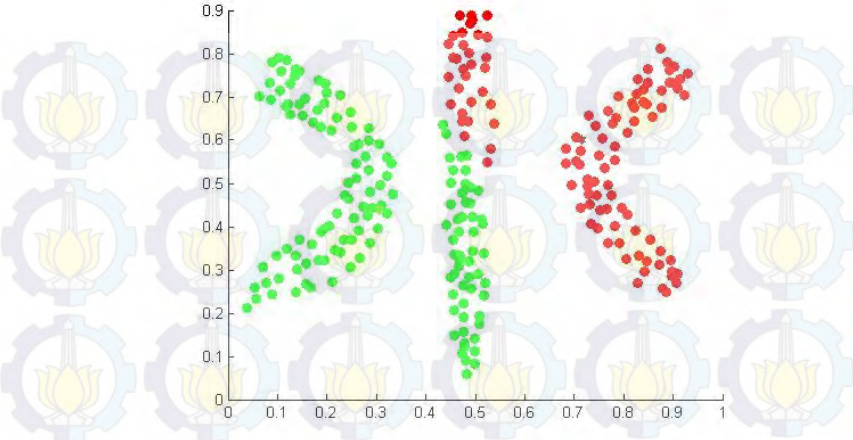
Gambar A.18. Hasil Uji Coba 1 menggunakan Kmeans pada percobaan ke-1



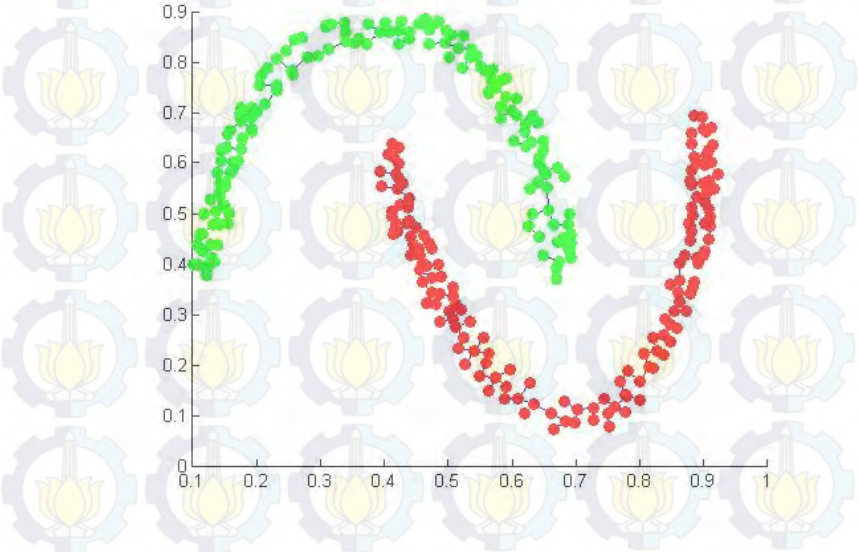
Gambar A.19. Hasil Uji Coba 1 menggunakan FMST pada percobaan ke-2



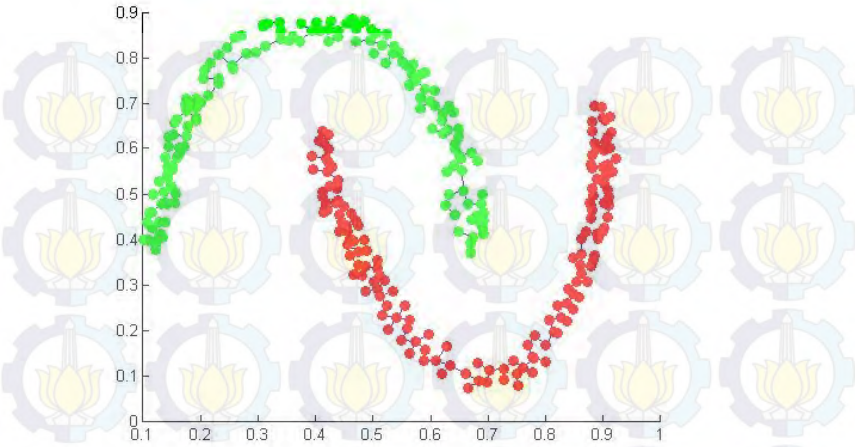
Gambar A.20. Hasil Uji Coba 1 menggunakan MST pada percobaan ke-2



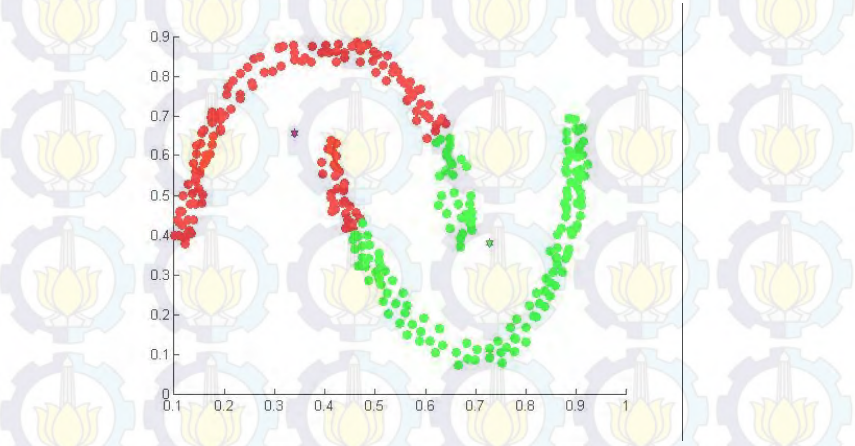
Gambar A.21. Hasil Uji Coba 1 menggunakan Kmeans pada percobaan ke-2



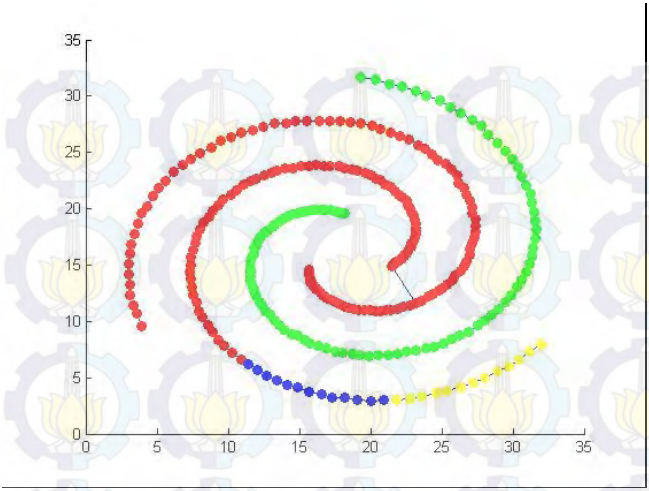
Gambar A.22. Hasil Uji Coba 1 menggunakan FMST pada percobaan ke-3



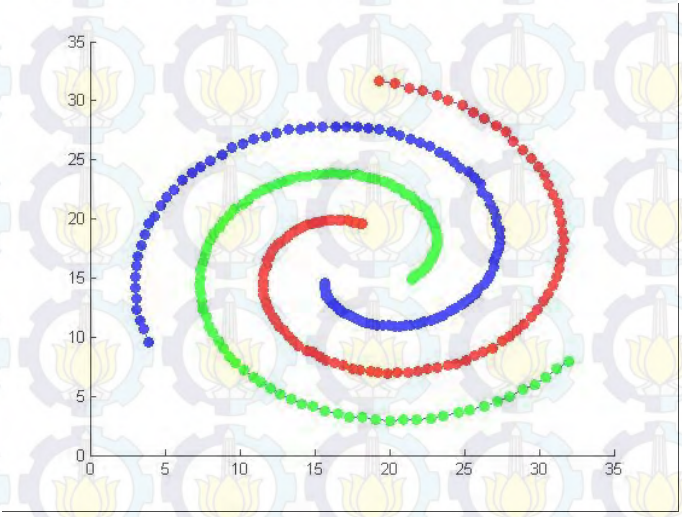
Gambar A.23. Hasil Uji Coba 1 menggunakan MST pada percobaan ke-3



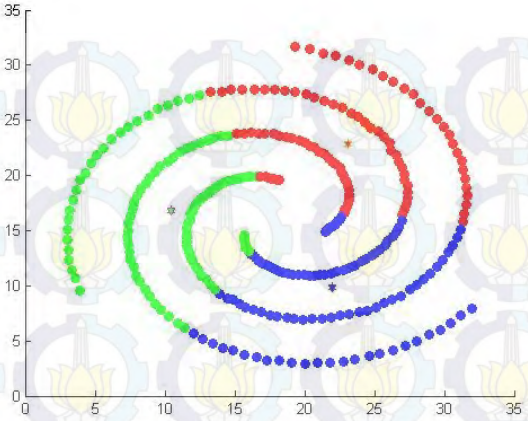
Gambar A.24. Hasil Uji Coba 1 menggunakan Kmeans pada percobaan ke-3



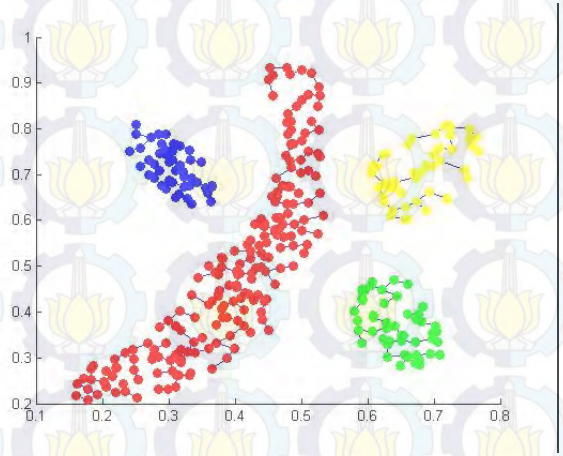
Gambar A.25. Hasil Uji Coba 1 menggunakan FMST pada percobaan ke-4



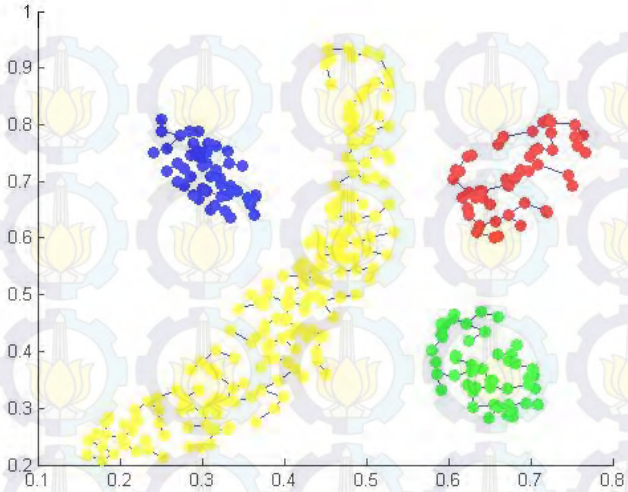
Gambar A.26 Hasil Uji Coba 1 menggunakan MST pada percobaan ke-4



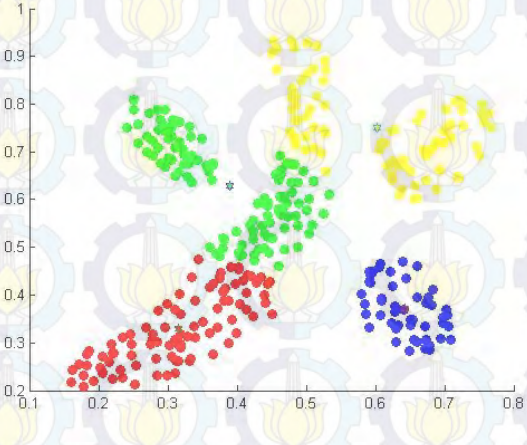
Gambar A.27. Hasil Uji Coba 1 menggunakan Kmeans pada percobaan ke-4



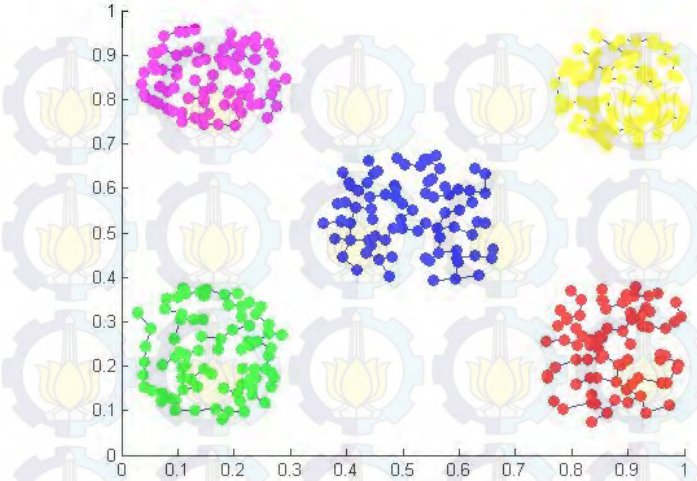
Gambar A.28. Hasil Uji Coba 1 menggunakan FMST pada percobaan ke-5



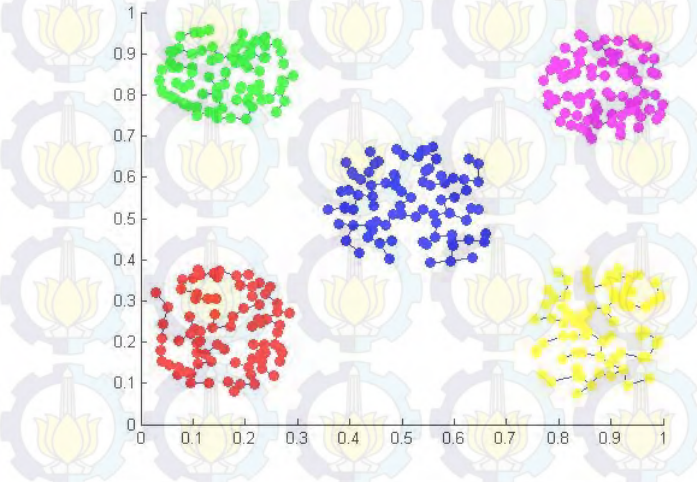
Gambar A.29. Hasil Uji Coba 1 menggunakan MST pada percobaan ke-5



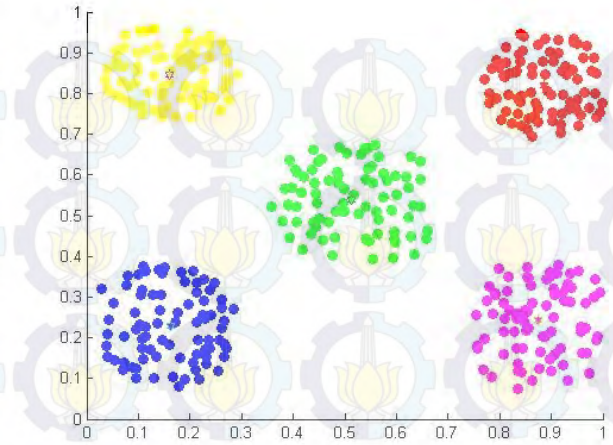
Gambar A.30. Hasil Uji Coba 1 menggunakan Kmeans pada percobaan ke-5



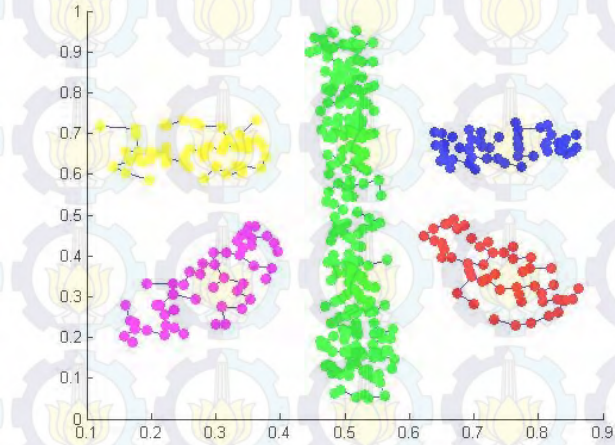
Gambar A.31. Hasil Uji Coba 1 menggunakan FMST pada percobaan ke-6



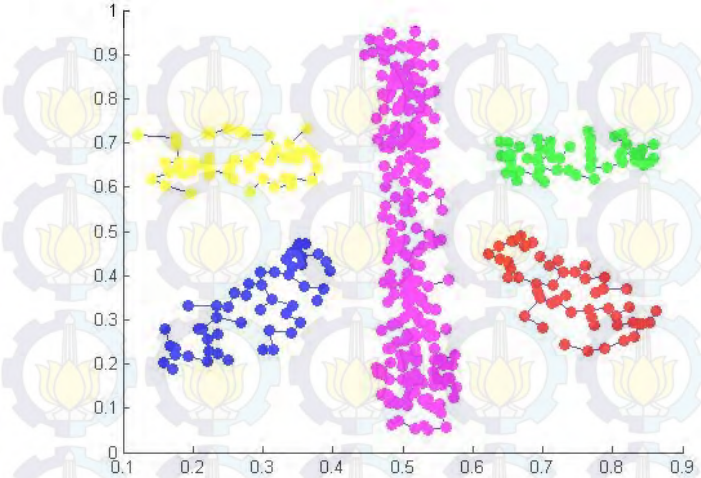
Gambar A.32. Hasil Uji Coba 1 menggunakan MST pada percobaan ke-6



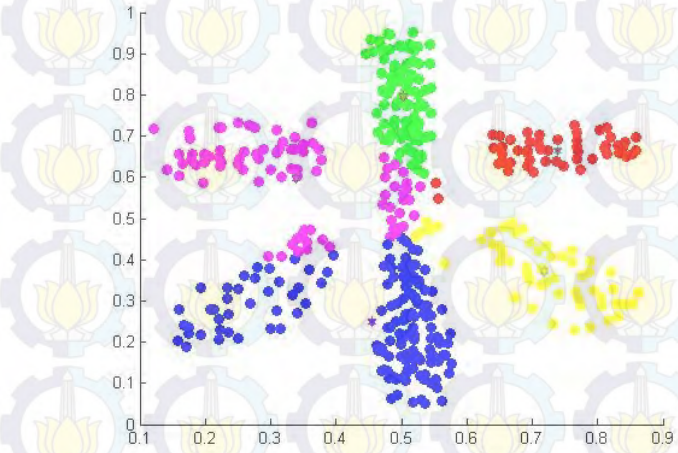
Gambar A.33. Hasil Uji Coba 1 menggunakan Kmeans pada percobaan ke-6



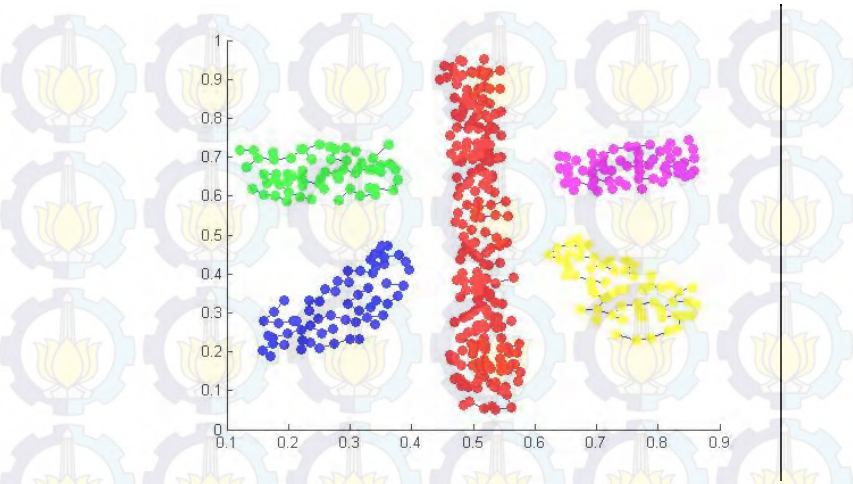
Gambar A.34. Hasil Uji Coba 1 menggunakan FMST pada percobaan ke-7



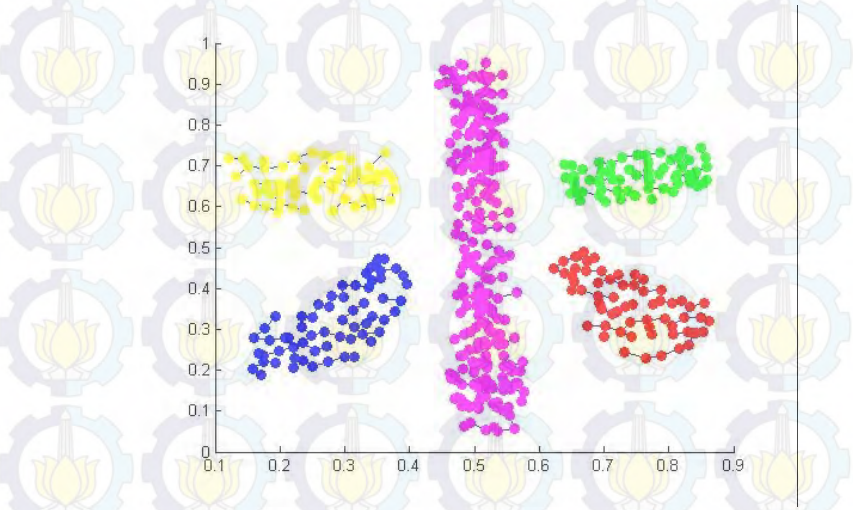
Gambar A.35. Hasil Uji Coba 1 menggunakan MST pada percobaan ke-7



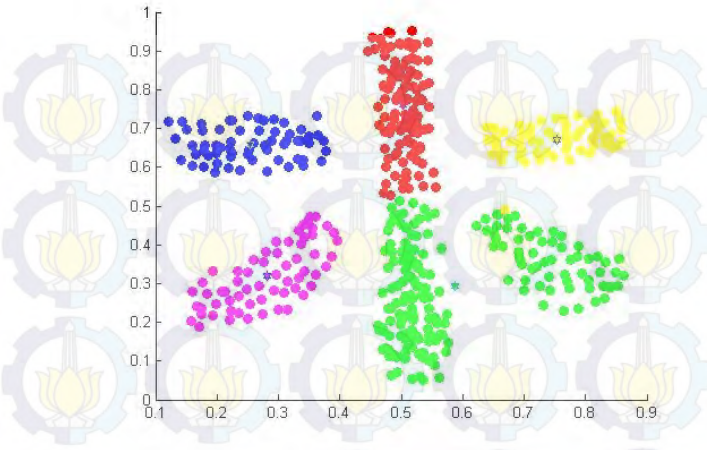
Gambar A.36. Hasil Uji Coba 1 menggunakan Kmeans pada percobaan ke-7



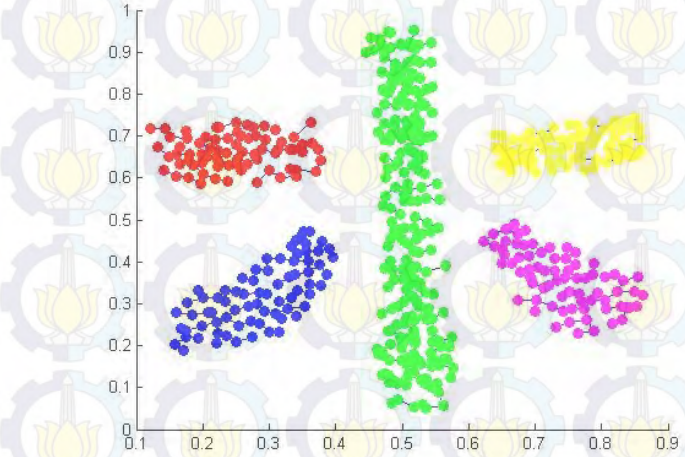
Gambar A.37. Hasil Uji Coba 1 menggunakan FMST pada percobaan ke-8



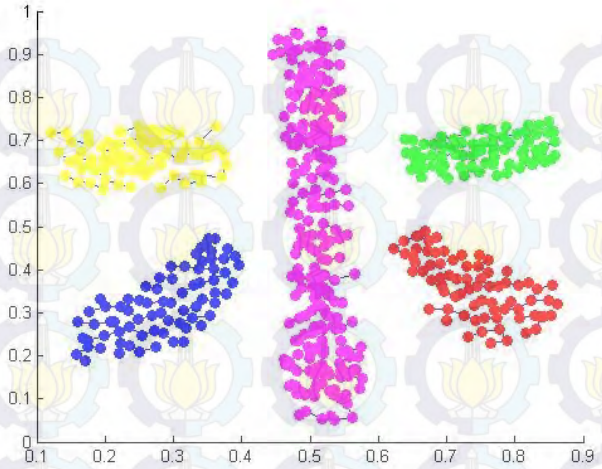
Gambar A.38. Hasil Uji Coba 1 menggunakan MST pada percobaan ke-8



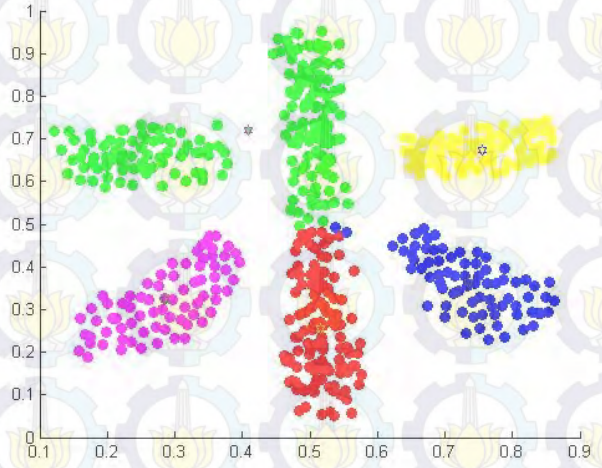
Gambar A.39. Hasil Uji Coba 1 menggunakan Kmeans pada percobaan ke-8



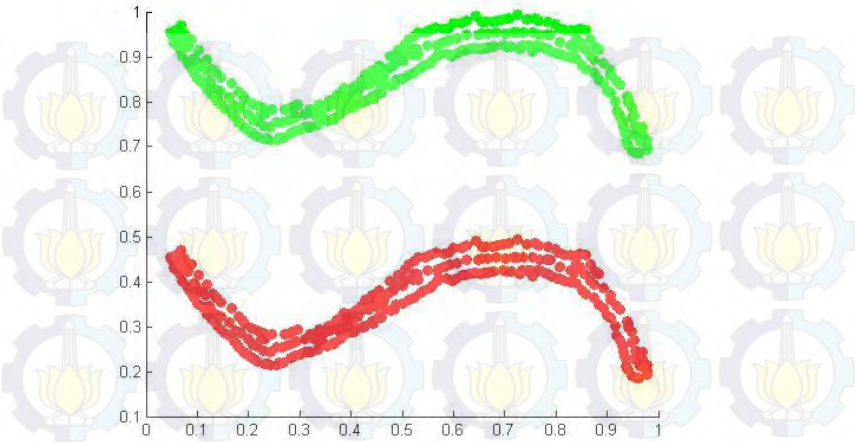
Gambar A.40. Hasil Uji Coba 1 menggunakan FMST pada percobaan ke-9



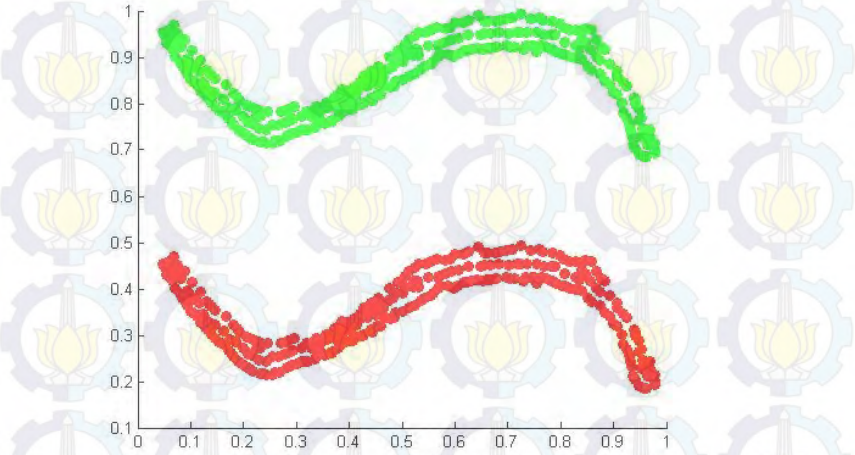
Gambar A.41. Hasil Uji Coba 1 menggunakan MST pada percobaan ke-9



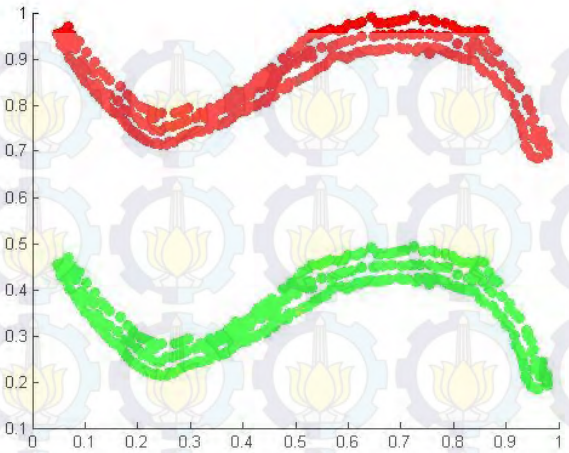
Gambar A.42. Hasil Uji Coba 1 menggunakan Kmeans pada percobaan ke-9



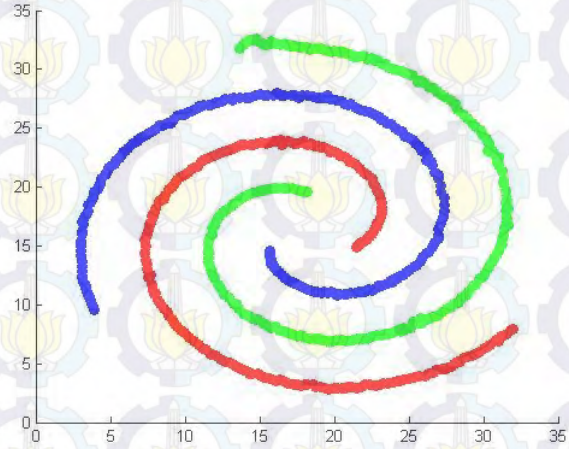
Gambar A.43. Hasil Uji Coba 1 menggunakan FMST pada percobaan ke-10



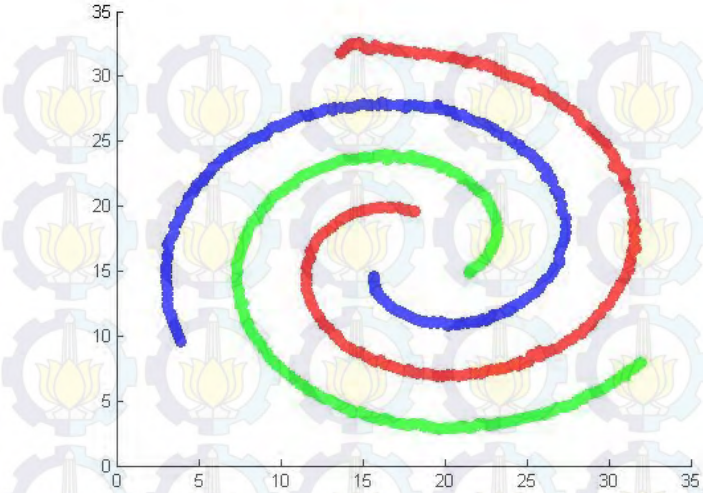
Gambar A.44. Hasil Uji Coba 1 menggunakan MST pada percobaan ke-10



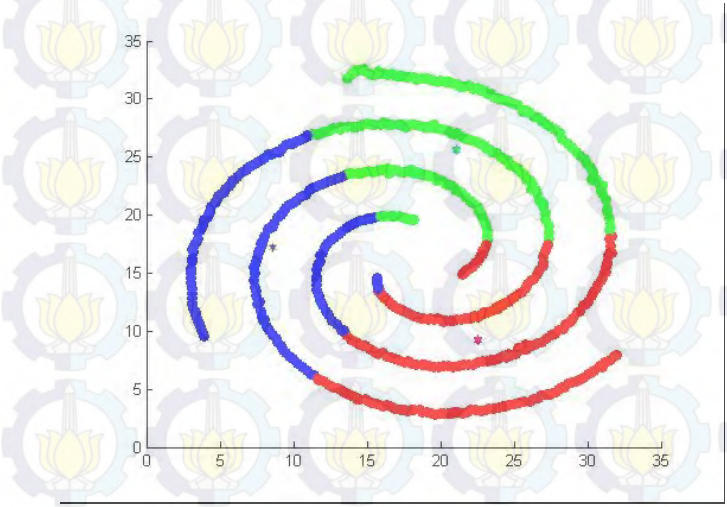
Gambar A.45. Hasil Uji Coba 1 menggunakan Kmeans pada percobaan ke-10



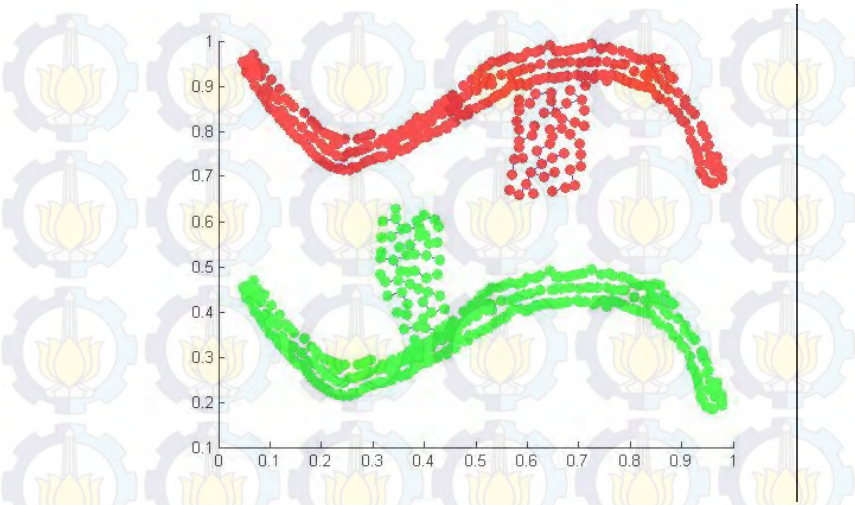
Gambar A.46. Hasil Uji Coba 1 menggunakan FMST pada percobaan ke-11



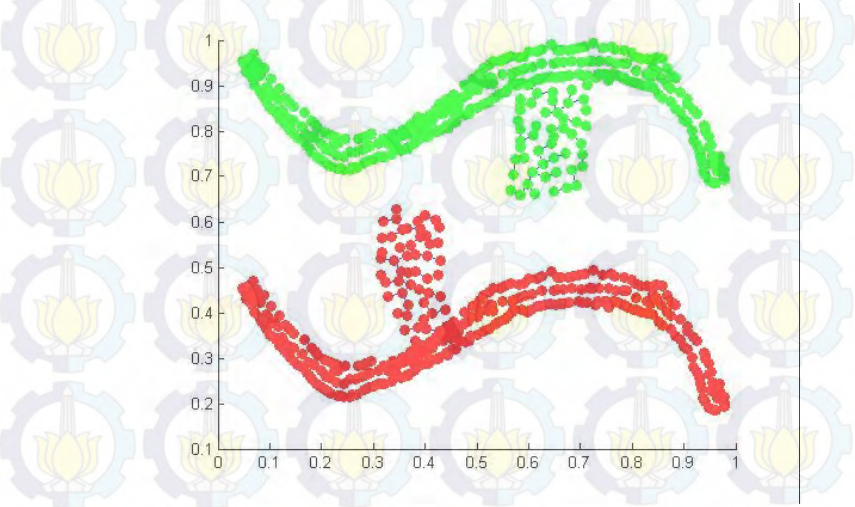
Gambar A.47. Hasil Uji Coba 1 menggunakan MST pada percobaan ke-11



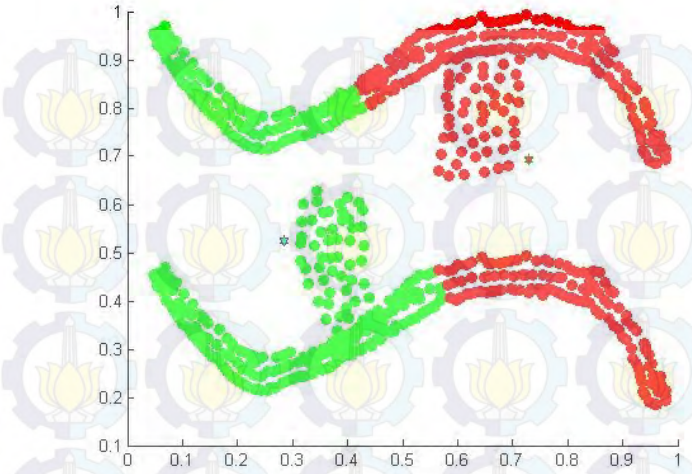
Gambar A.48. Hasil Uji Coba 1 menggunakan Kmeans pada percobaan ke-11



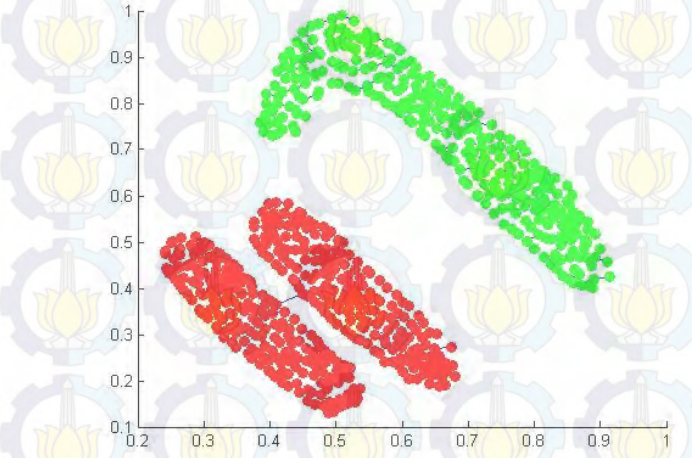
Gambar A.49. Hasil Uji Coba 1 menggunakan FMST pada percobaan ke-12



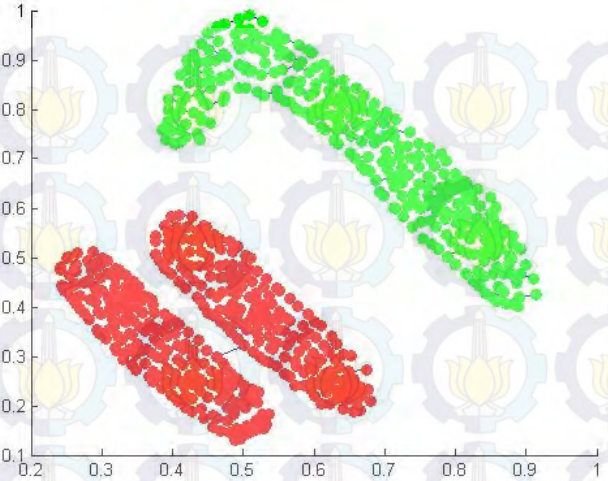
Gambar A.50. Hasil Uji Coba 1 menggunakan MST pada percobaan ke-12



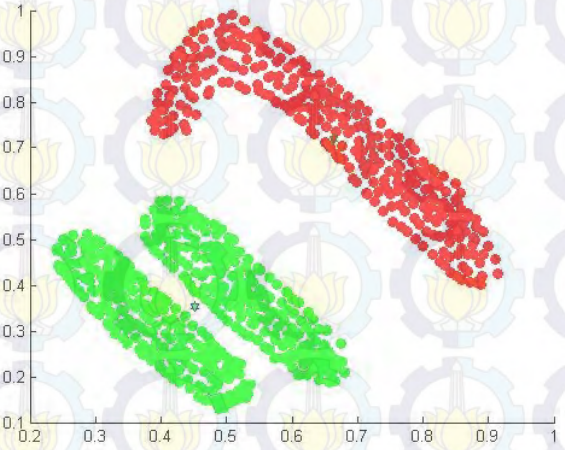
Gambar A.51. Hasil Uji Coba 1 menggunakan Kmeans pada percobaan ke-12



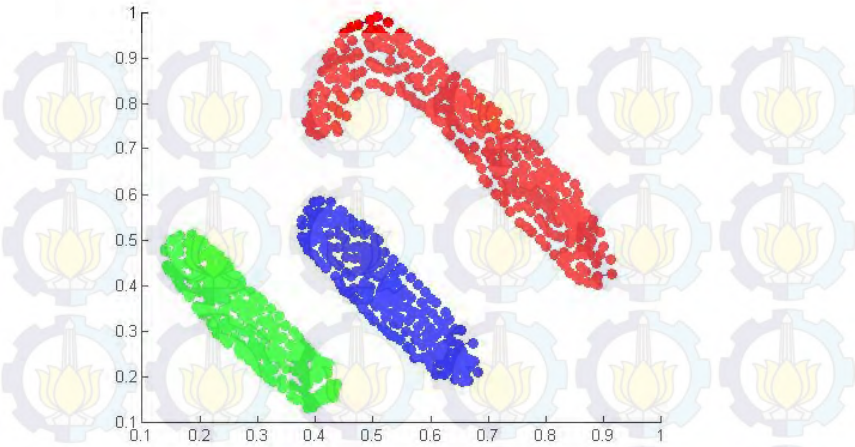
Gambar A.52. Hasil Uji Coba 1 menggunakan FMST pada percobaan ke-13



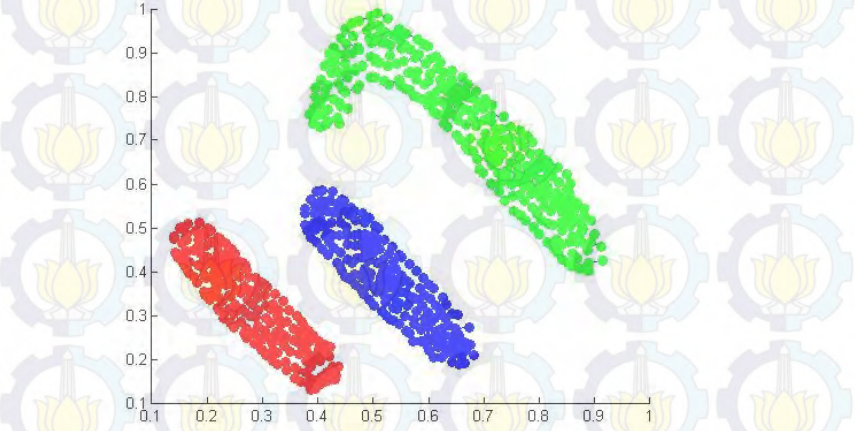
Gambar A.53. Hasil Uji Coba 1 menggunakan MST pada percobaan ke-13



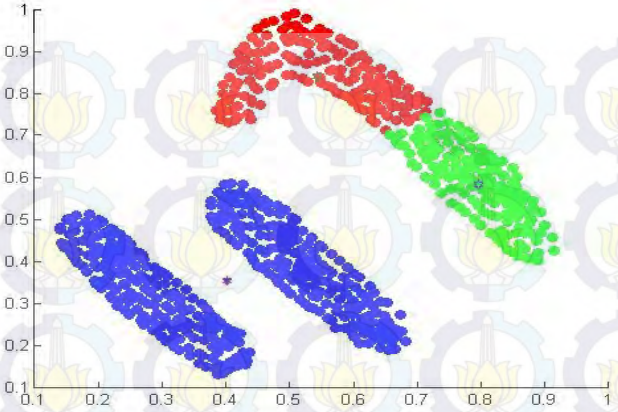
Gambar A.54. Hasil Uji Coba 1 menggunakan Kmeans pada percobaan ke-13



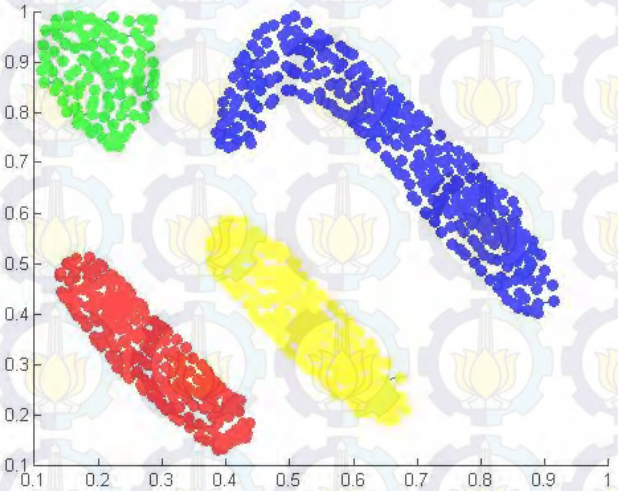
Gambar A.55. Hasil Uji Coba 1 menggunakan FMST pada percobaan ke-14



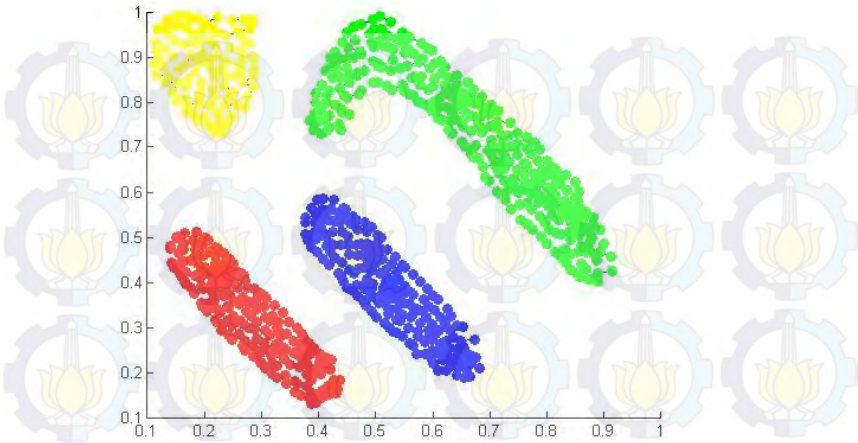
Gambar A.56. Hasil Uji Coba 1 menggunakan FMST pada percobaan ke-14



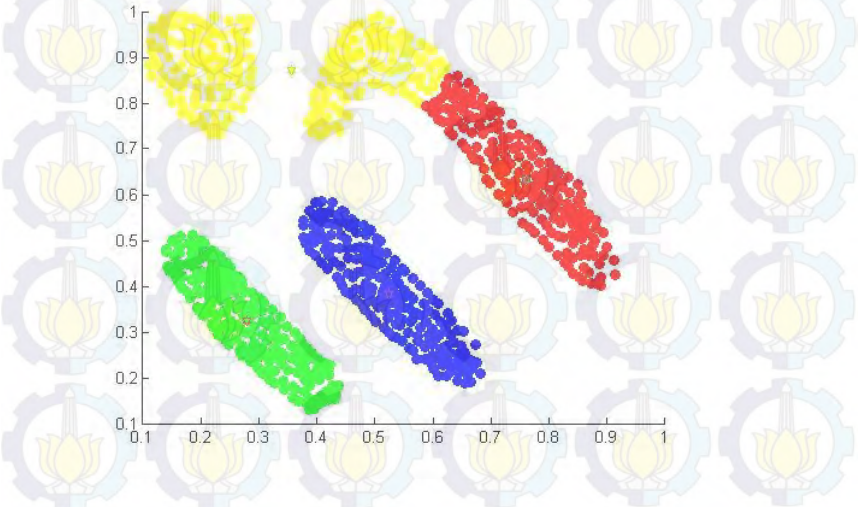
Gambar A.57. Hasil Uji Coba 1 menggunakan Kmeans pada percobaan ke-14



Gambar A.58. Hasil Uji Coba 1 menggunakan FMST pada percobaan ke-15



Gambar A.59. Hasil Uji Coba 1 menggunakan MST pada percobaan ke-15



Gambar A.60. Hasil Uji Coba 1 menggunakan Kmeans pada percobaan ke-15

UNDERGRADUATE THESES – KI141502

IMPLEMENTATION OF FAST MINIMUM SPANNING TREE FOR CLUSTERING ON PATTERN RECOGNITION

STEVEN FREDIAN ANDY PUTRA
NRP 5111100060

Supervisor I
Dr. Eng. Chastine Fatichah, S.Kom., M.Kom.

Supervisor II
Diana Purwitasari, S.Kom, M.Sc.

DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA 2015

LEMBAR PENGESAHAN

IMPLEMENTASI FAST MINIMUM SPANNING TREE UNTUK MELAKUKAN PENGELOMPOKAN DATA PADA PENGENALAN POLA

TUGAS AKHIR

Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Rumpun Mata Kuliah Komputasi Cerdas dan Visi
Program Studi S-1 Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember

Oleh
STEVEN FREDIAN A.P.
NRP : 5111100060

Disetujui oleh Pembimbing Tugas Akhir:

1. Dr. Eng. Chastine Fatichah, S.Kom., M.Kom. .....
   NIP: 19751220 200112 2 002 (Pembimbing1)

2. Diana Purwitasari, S.Kom., M.Sc. .....
   NIP: 19780410 200312 2 001 (Pembimbing 2)

SURABAYA
JUNI, 2015

IMPLEMENTASI *FAST MINIMUM SPANNING TREE* UNTUK MELAKUKAN PENGELOMPOKAN DATA PADA PENGENALAN POLA

Nama Mahasiswa : STEVEN FREDIAN ANDY PUTRA
NRP : 5111100060
Jurusan : Teknik Informatika FTIF-ITS
Dosen Pembimbing 1 : Dr.Eng. Chastine Fatichah, S.Kom., M.Kom.
Dosen Pembimbing 2 : Diana Purwitasari, S.Kom , M.Sc

Abstrak

Pengelompokan data berdasarkan kemiripan pola untuk data non-convex tidak bisa dilakukan dengan algoritma klastering biasa seperti k-means. Data bersifat convex memiliki kondisi jika diambil sembarang pasangan data maka garis yang menghubungkan data termasuk dalam himpunan tersebut. Minimum Spanning Tree (MST) dapat digunakan untuk menyelesaikan permasalahan pengelompokan data non-convex. Hal ini dikarenakan MST merupakan pohon rentang dari suatu graph yang menghubungkan semua titik dengan bobot paling minimal. Akan tetapi implementasi MST konvensional ke dataset yang besar membutuhkan banyak waktu.

Pada Tugas Akhir ini dibentuk solusi untuk menyelesaikan pengelompokan data dengan metode MST yang dimodifikasi. Solusi tersebut mengombinasikan metode Fast Minimum Spanning Tree (FMST) dan klastering berdasarkan MST. FMST menghasilkan MST-perkiraan dengan waktu yang lebih cepat. FMST menggunakan konsep divide and conquer (data dibagi menjadi beberapa kelompok) lalu diselesaikan MST tiap kelompok, dan semuanya digabungkan. Klastering berdasarkan MST menghapus edge yang memiliki bobot melebihi dari threshold yang ditentukan. Pada setiap

perulangan menambah threshold sampai tidak ada edge yang dapat dihapus, dilakukan uji validitas indeks Ray & Turi untuk mengetahui hasil klastering paling optimal.

Uji coba dilakukan pada data sintesis yang telah disiapkan serta data iris dan wine. Hasil uji coba menunjukkan bahwa metode FMST mampu mencari MST pada dataset dengan waktu lebih cepat. Perbedaan jumlah bobot edge pada FMST dengan MST sebenarnya atau weight error rata-rata yang dihasilkan FMST tidak melebihi 1 persen. Klastering FMST mampu mengidentifikasi jumlah klaster dengan benar pada 13 dari 15 uji coba data sintesis yang tidak bersifat convex.

Kata kunci: Pengenalan Pola, Klastering, Convex Set, Minimum Spanning Tree, Fast Minimum Spanning Tree, Indeks Dunn.

IMPLEMENTATION OF FAST MINIMUM SPANNING TREE FOR CLUSTERING ON PATTERN RECOGNITION

Student's Name	**:**	**STEVEN FREDIAN ANDY PUTRA**
Student's ID	**:**	**5111100060**
Department	**:**	**Teknik Informatika FTIF-ITS**
First Advisor	**:**	**Dr.Eng. Chastine Fatichah, S.Kom., M.Kom.**
Second Advisor	**:**	**Diana Purwitasari, S.Kom , M.Sc**

Abstract

Grouping data based on their pattern similarities for non-convex data cannot be done with typical clustering algorithms such as k-means. Convex data is defined with certain characteristic so that a pair of data items could have a connected line which also consists data items included in the dataset. Minimum Spanning Tree (MST) can be used to solve problem of grouping non-convex data. MST is a tree that connects data items with the most minimal weight value. However conventional MST requires a great deal of running time.

A modified MST is implemented in this work called Fast Minimum Spanning Tree (FMST) which combines MST and a clustering method based on MST. FMST could result an approximation of MST outcome with faster running time. FMST used an approach of divide-conquer that divides data into several groups, resolved the MST of each group, and combined all of those MST groups to get the final MST. The MST-based clustering in FMST removes edges with weight value that exceeds a specified threshold value. Ray & Turi index value is used to determine the most optimal clustering

results in each iteration process for adjusting the threshold value until there are no more edges that can be removed.

The experiments used synthetic data which has been prepared beforehand as well as standard data (Iris and Wine dataset). The experiment indicator is weight error value which means the differences in the summation value of edge weight for FMST result and the real MST result. The experiment in this work showed less than one percent weight error value. FMST could identify cluster numbers correctly in 13 of the 15 trials of the synthetic data.

Keywords: Pattern Recognition, Clustering, Convex Set, Minimum Spanning Tree, Fast Minimum Spanning Tree, Index Dunn

KATA PENGANTAR

Segala puji dan syukur kepada Tuhan Yang Maha Esa yang telah memberikan rahmat dan karunia-Nya sehingga penulis dapat menyelesaikan tugas akhir yang berjudul "*Implementasi Fast Minimum Spanning Tree untuk Melakukan Pengelompokan Data pada Pengenalan Pola*".

Tugas Akhir ini diajukan untuk memenuhi salah satu syarat memperoleh gelar Sarjana Komputer di Jurusan Teknik Informatika Fakultas Teknologi Informasi Institut Teknologi Sepuluh Nopember.

Dengan selesainya Tugas Akhir ini, penulis berharap apa yang penulis telah kerjakan dapat memberikan manfaat bagi perkembangan ilmu pengetahuan serta bagi penulis.

Keberhasilan dalam penyelesaian tugas akhir ini tidak terlepas dari bantuan berbagai pihak. Maka dari itu, pada kesempatan ini penulis ingin menyampaikan rasa terima kasih yang tulus dan sebesar-besarnya kepada semua pihak, terutama kepada:

1. Tuhan Yang Maha Esa yang telah memberikan rahmat dan karunia-Nya sehingga penulis dapat menyelesaikan Tugas Akhir dengan baik.
2. Keluarga penulis (Papa Andy, Mama Ririn, dan adik Pipin) yang senantiasa memberikan doa dan dukungan baik secara moral maupun finansial selama penulis menyelesaikan Tugas Akhir.
3. Bapak Dr. Agus Zainal Arifin, S.Kom, M.Kom selaku dosen wali penulis, yang selalu memberikan bimbingan, dukungan, serta motivasi selama masa perkuliahan di Teknik Informatika ITS.
4. Ibu Dr.Eng. Chastine Fatichah, S.Kom., M.Kom., selaku dosen pembimbing I atas bimbingan, ilmu, kesabaran dan bantuan-bantuan berharga lainnya sehingga penulis dapat menyelesaikan tugas akhir ini tepat pada waktunya.

5. Ibu Diana Purwitasari, S.Kom, M.Sc. selaku dosen pembimbing II atas bimbingan, bantuan dan kesabaran selama proses pengerjaan tugas akhir.
6. Bapak dan Ibu dosen dan karyawan jurusan Teknik Informatika ITS yang telah banyak mengajarkan ilmu pada penulis selama menempuh studi di Teknik Informatika ITS.
7. Seluruh teman penulis, khususnya teman-teman Angkatan 2011 dalam menempuh perkuliahan di Jurusan Teknik Informatika Fakultas Teknologi Informasi Institut Teknologi Sepuluh Nopember.
8. Dan seluruh pihak yang turut membantu dalam pengerjaan tugas akhir ini, baik secara moril maupun materil, yang tidak dapat penulis sebutkan satu per satu di sini.

Penulis berharap bahwa tugas akhir ini dapat memberikan manfaat pada semua pihak khususnya penulis sendiri dan *civitas academica* Teknik Informatika ITS. Penulis mohon maaf bila terdapat kesalahan, kelalaian maupun kekurangan dalam penyusunan tugas akhir ini. Kritik dan saran yang membangun dapat disampaikan sebagai bahan perbaikan kedepan.

Surabaya, Juni 2015
Penulis

DAFTAR ISI

[Halaman ini sengaja dikosongkan]

DAFTAR TABEL

[Halaman ini sengaja dikosongkan]

DAFTAR GAMBAR

DAFTAR KODE SUMBER

[Halaman ini sengaja dikosongkan]

BIODATA PENULIS

Steven Fredian Andy Putra, lahir di Pasuruan, pada tanggal 1 April 1994. Penulis menempuh pendidikan mulai dari SDK Sang Timur Pasuruan (1999-2005), SMPK Sang Timur Pasuruan (2005-2008), SMAN 1 Pasuruan (2008-2011) dan S1 Teknik Informatika ITS (2011-2015).

Selama masa kuliah, penulis aktif dalam organisasi Himpunan Mahasiswa Teknik Computer (HMTC). Diantaranya adalah menjadi staff departemen Dalam Negeri Himpunan Mahasiswa Teknik Computer ITS 2012-2013 dan staff ahli Dalam Negeri Himpunan Mahasiswa Teknik Computer ITS 2013-2014. Penulis juga aktif dalam kegiatan kepanitiaan Schematics. Diantaranya penulis pernah menjadi sttaff seminar nasional NST Schematics 2012 dan staff Humas Schematics 2013.

Selama kuliah di teknik informatika ITS, penulis mengambil bidang minat Komputasi Cerdas Visual (KCV). Komunikasi dengan penulis dapat melalui email: **steven.fredian@gmail.com**.

BAB I
PENDAHULUAN

1.1 Latar Belakang

Pengenalan pola adalah suatu kemampuan untuk mengelompokkan obyek berdasarkan parameter-parameter yang telah ditentukan kedalam sejumlah kategori atau kelas. Pengenalan pola dapat dibagi menjadi dua, yaitu *supervised*, dan *unsupervised*. *Supervised* adalah pembelajaran yang memiliki suatu contoh, sehingga hasil dari pembelajaran ini mengacu pada contoh yang diberikan. Sebaliknya, *unsupervised* adalah pembelajaran yang tidak memiliki contoh, sehingga hasil dari pembelajaran ini menggunakan prosedur yang berusaha untuk mencari partisi dari sebuah pola [1]. Salah satu implementasi *unsupervised learning* adalah klastering.

Klastering merupakan suatu proses pembagian dari suatu kumpulan data menjadi kelompok – kelompok yang lebih kecil berdasarkan kesamaan fitur [2]. Klastering memegang peranan penting di beberapa bidang, seperti *image processing*, komputasi biomedik, ekonomi, dan lain- lain. Sebagai salah satu contoh di bidang *image processing*, klastering dapat digunakan untuk segmentasi citra berdasarkan warna [3]. Citra dapat dibagi ke beberapa kelompok berdasarkan kemiripan warna di dalamnya.

Kemajuan teknologi saat ini menyebabkan berkembangnya data – data secara pesat. Menurut macamnya, data dibagi menjadi dua, yaitu data real, dan data sintesis. Data sintesis merupakan data buatan yang didesain mengikuti suatu kondisi tertentu yang tidak dapat ditemukan di data real [4]. Dalam hal ini, data sintesis dibentuk menyerupai pola seperti spiral, curve, dan lain-lain. Data tersebut disebut *Convex Set.* Algoritma klastering yang menggunakan konsep jarak terdekat dengan *centroid* seperti K-Means tidak dapat menyelesaikan masalah pengelompokan data yang bersifat *non-Convex* [5].

Untuk mengatasi masalah tersebut, maka akan digunakan klastering dengan menggunakan *Minimum Spanning Tree* (*MST*). Akan tetapi, sulit untuk mengimplementasikan algoritma konvensional untuk menyelesaikan permasalahan *MST* ke *dataset* yang besar, dikarenakan kompleksitas yang tinggi [6]. Maka dari itu, akan digunakan *Fast Minimum Spanning Tree* (*FMST*) untuk mendapatkan *MST* dengan waktu lebih singkat. *FMST* menggunakan teknik *Divide and Conquer*, dimana *dataset* dibagi ke beberapa *subset* menggunakan K-Means, lalu diselesaikan permasalahan *MST* nya tiap *subset* terlebih dahulu. Kemudian, *MST* tiap *subset* dihubungkan untuk membentuk suatu *MST*. *Set*elah didapatkan hasil *FMST* dari suatu data, klastering dilakukan dengan cara menghapus *edge* dari *FMST* yang bernilai lebih besar dari *threshold*.

Hasil klastering menggunakan *FMST* ini diperoleh dengan waktu yang lebih singkat dan mencapai akurasi yang baik.

1.2 Rumusan Masalah

Rumusan masalah yang diangkat dalam Tugas Akhir ini dapat dipaparkan sebagai berikut:

1. Bagaimana melakukan pengelompokan data menggunakan *FMST*?
2. Bagaimana mengevaluasi kinerja dari pengelompokan data menggunakan *FMST*?

1.3 Batasan Masalah

Permasalahan yang dibahas dalam Tugas Akhir ini memiliki beberapa batasan, yaitu sebagai berikut:

1. Data yang digunakan merupakan data sintesis (2 dimensi/fitur) dan *dataset* lain (*iris, wine)* bersifat numerik.
2. Algoritma *MST* yang digunakan adalah Algoritma Prim.
3. Algoritma *FMST* yang digunakan adalah metode yang diusulkan Caiming Zhong, dkk [6].

4. Algoritma Klastering berdasarkan *MST* yang digunakan adalah algoritma yang diusulkan oleh Prasanta Jana dan Azad Naik [7].

1.4 Tujuan

Tugas akhir ini adalah mengimplementasikan *Fast Minimum Spanning Tree* untuk melakukan pengelompokan data.

1.5 Manfaat

Manfaat yang dapat diambil dari pengerjaan tugas akhir ini yaitu dapat menyelesaikan pengelompokan data yang bersifat *non-Convex* dengan baik.

1.6 Metodologi

Tahapan-tahapan yang dilakukan dalam pengerjaan Tugas Akhir ini adalah sebagai berikut:

1. Penyusunan proposal Tugas Akhir.
   Tahap awal untuk memulai pengerjaan Tugas Akhir adalah penyusunan proposal Tugas Akhir. Proposal Tugas Akhir yang diajukan memiliki gagasan yang sama dengan Tugas Akhir ini, yaitu implementasi metode *FMST, dan* klastering berdasarkan *MST*.

2. Studi literatur
   Pada tahap ini dilakukan pencarian, pengumpulan, pembelajaran dan pemahaman informasi dan literatur yang diperlukan untuk pembuatan implementasi metode *FMST* dan klastering berdasarkan *MST*. Informasi dan literatur didapatkan dari literatur buku dan sumber-sumber informasi lain yang berhubungan.

3. Analisis dan desain perangkat lunak

   Tahap ini meliputi perancangan sistem berdasarkan studi literatur dan pembelajaran konsep teknologi dari perangkat lunak yang ada. Tahap ini mendefinisikan alur dari implementasi. Langkah-langkah yang dikerjakan juga didefinisikan pada tahap ini. Pada tahapan ini dibuat *prototype* sistem, yang merupakan rancangan dasar dari sistem yang akan dibuat. Serta dilakukan desain suatu sistem dan desain proses-proses yang ada.

4. Implementasi perangkat lunak

   Implementasi merupakan tahap membangun rancangan program yang telah dibuat. Pada tahapan ini merealisasikan apa yang terdapat pada tahapan sebelumnya, sehingga menjadi sebuah program yang sesuai dengan apa yang telah direncanakan.

5. Pengujian dan evaluasi

   Pada tahapan ini dilakukan uji coba pada data yang telah dikumpulkan. Pengujian dan evaluasi akan dilakukan dengan menggunakan MATLAB. Tahapan ini dimaksudkan untuk mengevaluasi kesesuaian data dan program serta mencari masalah yang mungkin timbul dan mengadakan perbaikan jika terdapat kesalahan.

6. Penyusunan buku Tugas Akhir.

   Pada tahapan ini disusun buku yang memuat dokumentasi mengenai pembuatan serta hasil dari implementasi perangkat lunak yang telah dibuat.

1.7 Sistematika Penulisan Laporan Tugas Akhir

Buku Tugas Akhir ini bertujuan untuk mendapatkan gambaran dari pengerjaan Tugas Akhir ini. Selain itu, diharapkan dapat berguna untuk pembaca yang tertarik untuk melakukan

pengembangan lebih lanjut. Secara garis besar, buku Tugas Akhir terdiri atas beberapa bagian seperti berikut ini:

Bab I Pendahuluan

Bab yang berisi mengenai latar belakang, tujuan, dan manfaat dari pembuatan Tugas Akhir. Selain itu permasalahan, batasan masalah, metodologi yang digunakan, dan sistematika penulisan juga merupakan bagian dari bab ini.

Bab II Dasar Teori

Bab ini berisi penjelasan secara detail mengenai dasar-dasar penunjang dan teori-teori yang digunakan untuk mendukung pembuatan Tugas Akhir ini.

Bab III Perancangan Perangkat Lunak

Bab ini berisi tentang desain sistem yang disajikan dalam bentuk *pseudocode*.

Bab IV Implementasi

Bab ini membahas implementasi dari desain yang telah dibuat pada bab sebelumnya. Penjelasan berupa *code* yang digunakan untuk proses implementasi.

Bab V Uji Coba Dan Evaluasi

Bab ini menjelaskan kemampuan perangkat lunak dengan melakukan pengujian kebenaran dan pengujian kinerja dari sistem yang telah dibuat.

Bab VI Kesimpulan Dan Saran

Bab ini merupakan bab terakhir yang menyampaikan kesimpulan dari analisa hasil uji coba yang dilakukan

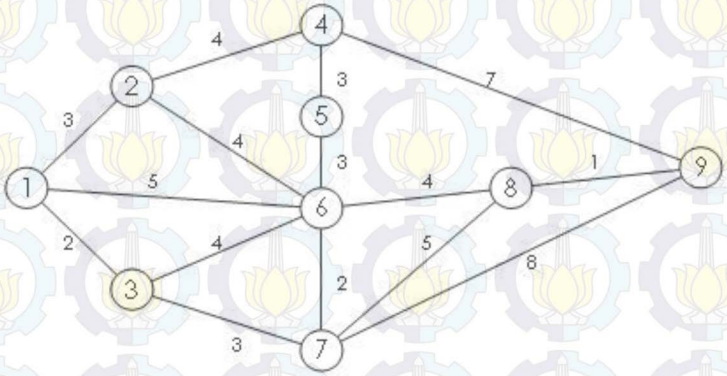
dan saran untuk pengembangan perangkat lunak ke depannya.

BAB II
TINJAUAN PUSTAKA

Bab ini berisi penjelasan teori-teori yang berkaitan dengan algoritma yang diajukan pada pengimplementasian program. Penjelasan ini bertujuan untuk memberikan gambaran secara umum terhadap program yang dibuat dan berguna sebagai penunjang dalam pengembangan perangkat lunak.

2.1 *Graph*

Graph G = (V, E) merupakan himpunan yang terdiri dari *vertex V* dan *edge E*. *Edge (u, v)* merupakan *edge* yang menghubungkan *vertex u* dan *vertex* v [8]. Jika *edge* pada *graph* memiliki bobot (berupa bilangan real, menandakan *cost, length*), maka disebut *weighted graph* [8]. *Graph* yang terdiri dari *edge* yang menghubungkan seluruh kombinasi pasangan data disebut *complete graph* [9]. Ilustrasi *graph* ditunjukkan pada Gambar 2.1 [10].

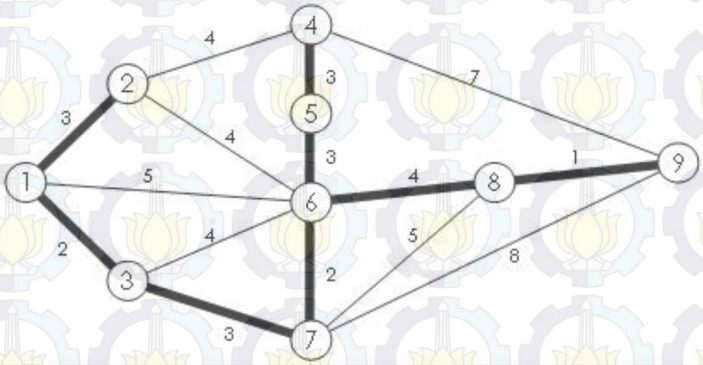


Gambar 2.1. Ilustrasi Graph

2.2 *Minimum Spanning Tree*

Minimum Spanning Tree (MST) adalah suatu *graph* yang memiliki batasan dimana semua *vertex* dalam *graph* terhubung tanpa terdapat *cycle* didalamnya dan memiliki total bobot *edge* yang paling minimum. *Cycle* merupakan rute dalam *graph* yang berawal dan berakhir pada *vertex* yang sama [11]. *Minimum Spanning Tree* banyak digunakan ketika ditemui masalah dalam hal penentuan seperti panjang kabel, khususnya dalam dunia kelistrikan, Jaringan (LAN), PDAM (penentuan panjangnya pipa) [12].

Sebagai contoh, Gambar 2.1 adalah denah saluran listrik pada suatu perusahaan. Teknisi listrik akan menyalurkan listrik dari ruang bagian depan sampai keseluruh ruangan dengan total panjang kabel yang seefisien mungkin. Ilustrasi *MST* dari Gambar 2.1 ditunjukkan pada Gambar 2.2 [10].

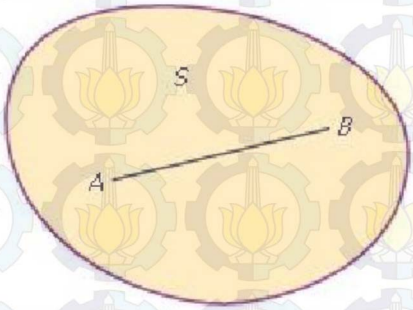


Gambar 2.2. Minimum Spanning Tree dari Gambar 2.1.

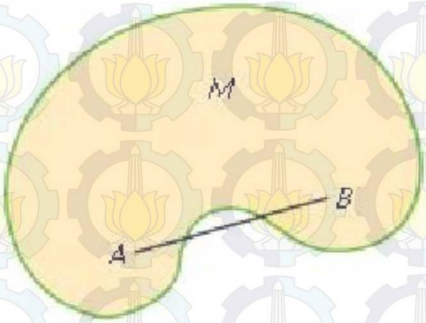
2.3 *Convex Set*

Convex Set adalah *set* atau himpunan titik, dimana setiap titik didalamnya dihubungkan oleh garis yang termasuk didalam *set* tersebut [13]. Pada Gambar 2.3 [14], ditunjukkan bahwa S

adalah contoh *Convex Set*. Titik A dan B pada S dihubungkan oleh garis yang termasuk dalam S. Sedangkan M pada Gambar 2.4 [14], merupakan contoh *Non-Convex Set*. A dan B pada M dihubungkan oleh garis yang tidak termasuk dalam M.



Gambar 2.3. Ilustrasi Convex Set



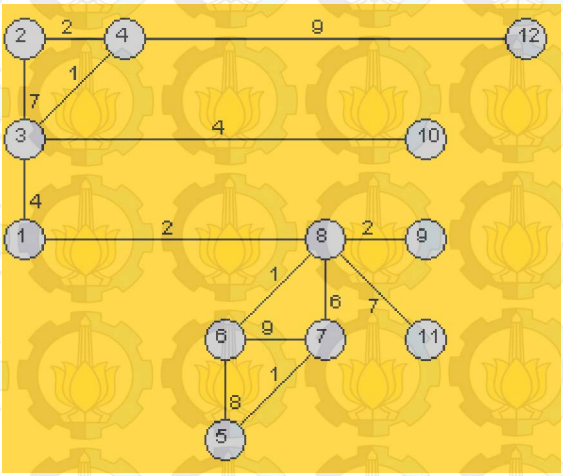
Gambar 2.4. Ilustrasi Non-Convex Set

2.4 Algoritma Prim

Algoritma Prim merupakan salah satu algoritma *greedy* yang dapat menyelesaikan permasalahan *MST* [15]. Diberikan *graph* G = (*U, V)*. Pada awalnya untuk setiap *vertex* $v \in V$

dilakukan inisialisasi *visited(v) = false*. Pilih salah satu *vertex s* $\in$ *V* secara bebas sebagai titik mulai. Set *visited(s) = true*. Kemudian enqueue setiap *edge(s, v)* $\in$ *V* beserta bobotnya pada *minimum weight priority queue D*. Selama *D* tidak kosong, cek *edge(u, s)* yang merupakan *top* dari *D*. *Dequeue top* dari *D*. Jika *visited(s) = false*, ubah *visited(s) = true*. Masukkan *edge(u,s)* kedalam himpunan *mst,* dan *enqueue* setiap *edge(s,v)* $\in$ *V* dengan syarat *visited(s)= false* pada *D*.

Pada akhir operasi, didapatkan himpunan berisi *list edge* yang merupakan *MST* dari *graph G*. Sebagai contoh, pada Gambar 2.5 divisualisasikan suatu *graph* yang ingin dicari *MST* nya dengan menggunakan Algoritma Prim.

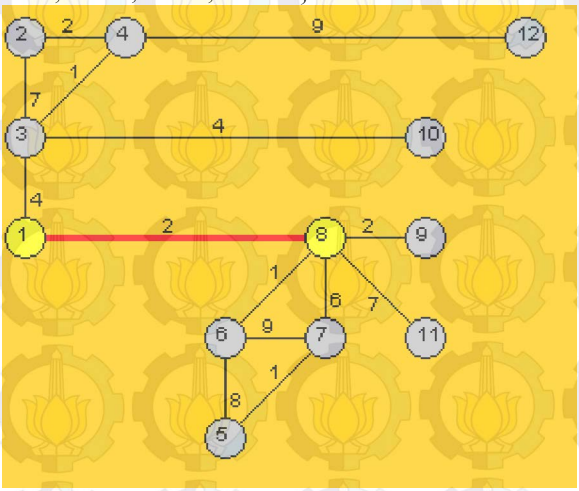


Gambar 2.5. Proses Algoritma Prim

Ditentukan *vertex* 1 sebagai *vertex* awal. Maka set *visited(*1*)= true*. Masukkan semua *edge* yang berawal dari *vertex 1* kedalam *minimum weight priority queue. Priority queue* saat ini berisi {1-8:2, 1-3:4}.

Pada *priority queue, top* saat ini adalah *edge(1,8)* dengan bobot 2. Setelah dicek *visited(*8*) = false*, maka set *visited(*8*) = true*, dan masukkan *edge(*1,8*)* ke dalam himpunan *mst. Dequeue top* dari

priority queue. Visualisasi sampai tahap ini dapat dilihat pada Gambar 2.6. Masukkan semua *edge* yang berawal dari *vertex* 8 ke *vertex* yang belum pernah dikunjungi. *Priority queue* saat ini berisi {8-6:1, 8-9:2, 1-3:4, 8-7:6, 8-11:7}.

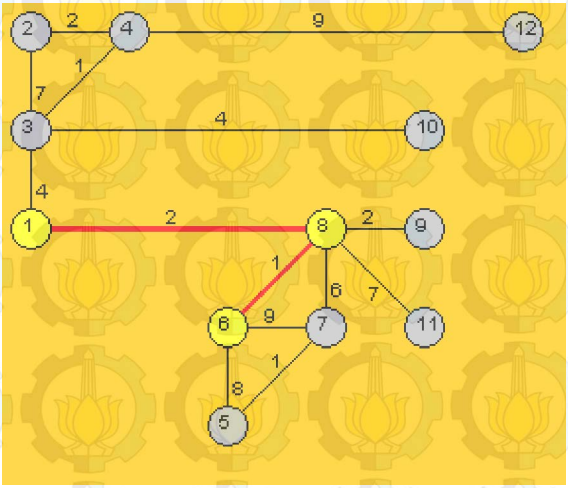


Gambar 2.6. Proses Algoritma Prim (2)

Pada *priority queue, top* saat ini adalah *edge(8,6)* dengan bobot 1. Setelah dicek *visited(*6*) = false*, maka set *visited(*6*) = true*, dan masukkan *edge(*8,6*)* ke dalam himpunan *mst. Dequeue top* dari *priority queue.* Visualisasi sampai tahap ini dapat dilihat pada Gambar 2.7. Masukkan semua *edge* yang berawal dari *vertex* 6 ke *vertex* yang belum pernah dikunjungi. *Priority queue* saat ini berisi {8-9:2, 1-3:4, 8-7:6, 8-11:7, 6-5:8, 6-7:9}.

Pada *priority queue, top* saat ini adalah *edge(8,9)* dengan bobot 2. Setelah dicek *visited(*9*) = false*, maka set *visited(*9*) = true*, dan masukkan *edge(*8,9*)* ke dalam himpunan *mst. Dequeue top* dari *priority queue.* Visualisasi sampai tahap ini dapat dilihat pada Gambar 2.8. Masukkan semua *edge* yang berawal dari *vertex* 9 ke *vertex* yang belum pernah dikunjungi. *Priority queue* saat ini berisi {1-3:4, 8-7:6, 8-11:7, 6-5:8, 6-7:9}.

Pada *priority queue, top* saat ini adalah *edge(1,3)* dengan bobot 4. Setelah dicek *visited(*4*) = false*, maka set *visited(*4*) = true*, dan masukkan *edge(*1,3*)* ke dalam himpunan *mst. Dequeue top* dari *priority queue.* Visualisasi sampai tahap ini dapat dilihat pada Gambar 2.9. Masukkan semua *edge* yang berawal dari *vertex* 3 ke *vertex* yang belum pernah dikunjungi. *Priority queue* saat ini berisi {3-4:1, 3-10:4, 8-7:6, 3-2:7, 8-11:7, 6-5:8, 6-7:9}.

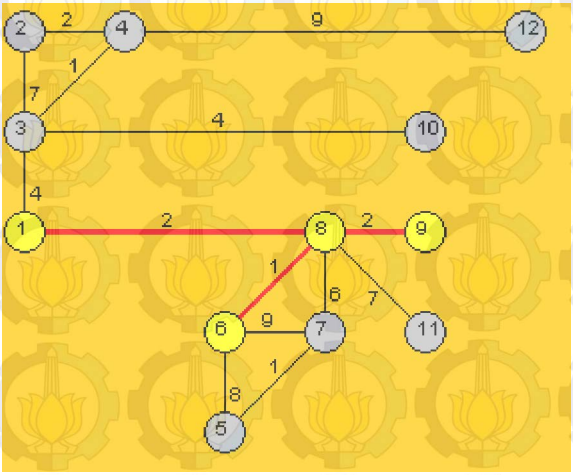


Gambar 2.7. Proses Algoritma Prim (3)

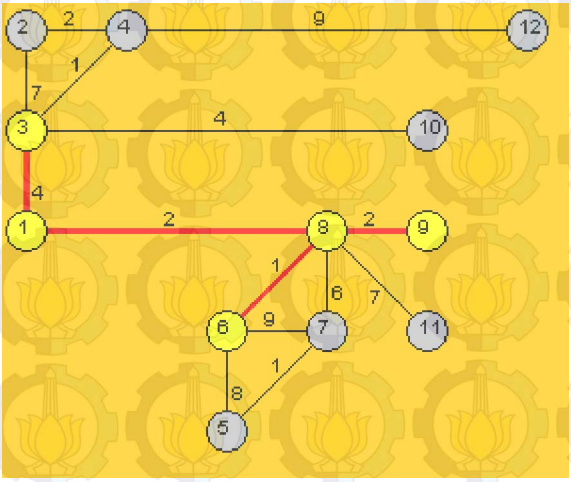
Pada *priority queue, top* saat ini adalah *edge(3,4)* dengan bobot 1. Setelah dicek *visited(*4*) = false*, maka set *visited(*4*) = true*, dan masukkan *edge(*3,4*)* ke dalam himpunan *mst. Dequeue top* dari *priority queue.* Visualisasi sampai tahap ini dapat dilihat pada Gambar 2.10. Masukkan semua *edge* yang berawal dari *vertex* 4 ke *vertex* yang belum pernah dikunjungi. *Priority queue* saat ini berisi {4-2:2, 3-10:4, 8-7:6, 3-2:7, 8-11:7, 6-5:8, 4-12:9, 6-7:9}.

Pada *priority queue, top* saat ini adalah *edge(4,2)* dengan bobot 2. Setelah dicek *visited(*2*) = false*, maka set *visited(*2*) = true*, dan masukkan *edge(*4,2*)* ke dalam himpunan *mst. Dequeue top* dari *priority queue.* Visualisasi sampai tahap ini dapat dilihat pada

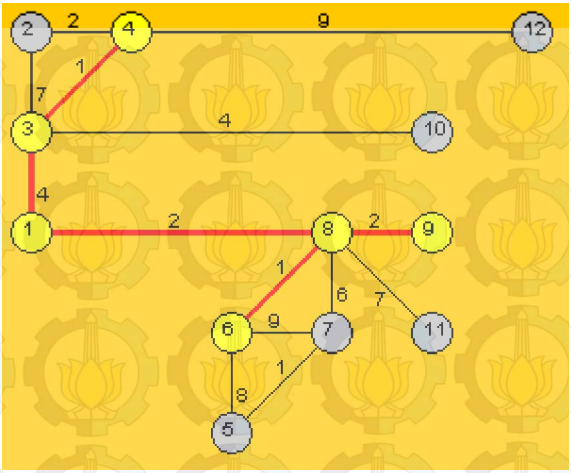
Gambar 2.11. Masukkan semua *edge* yang berawal dari *vertex* 2 ke *vertex* yang belum pernah dikunjungi. *Priority queue* saat ini berisi {3-10:4, 8-7:6, 3-2:7, 8-11:7, 6-5:8, 4-12:9, 6-7:9}.



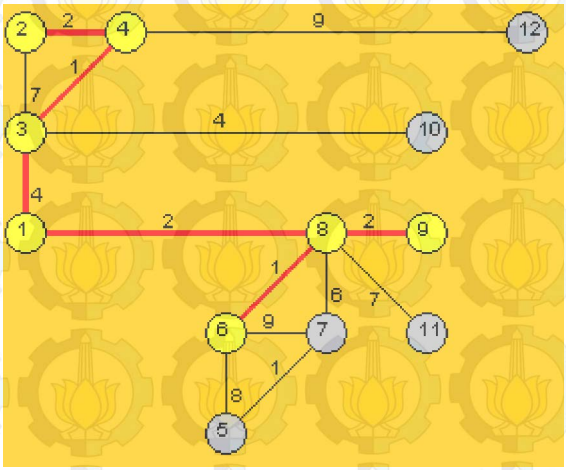
Gambar 2.8. Proses Algoritma Prim (4)



Gambar 2.9. Proses Algoritma Prim (5)



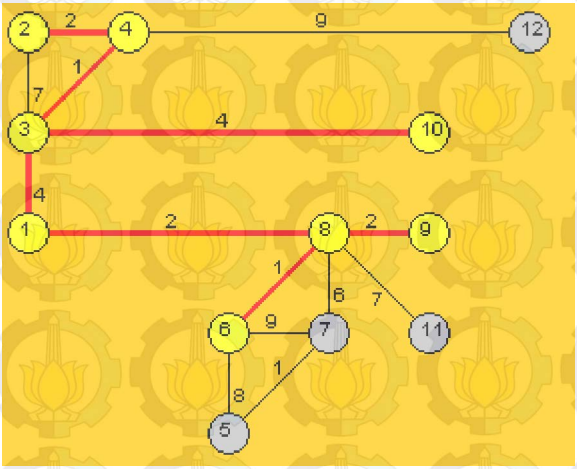
Gambar 2.10. Proses Algoritma Prim (6)



Gambar 2.11. Proses Algoritma Prim (7)

Pada *priority queue, top* saat ini adalah *edge(3,10)* dengan bobot 4. Setelah dicek *visited(*10*) = false*, maka set *visited(*10*) = true*, dan masukkan *edge(*3,10*)* ke dalam himpunan *mst. Dequeue*

top dari *priority queue.* Visualisasi sampai tahap ini dapat dilihat pada Gambar 2.12. Masukkan semua *edge* yang berawal dari *vertex* 10 ke *vertex* yang belum pernah dikunjungi. *Priority queue* saat ini berisi {8-7:6, 3-2:7, 8-11:7, 6-5:8, 4-12:9, 6-7:9}.



Gambar 2.12. Proses Algoritma Prim (8)

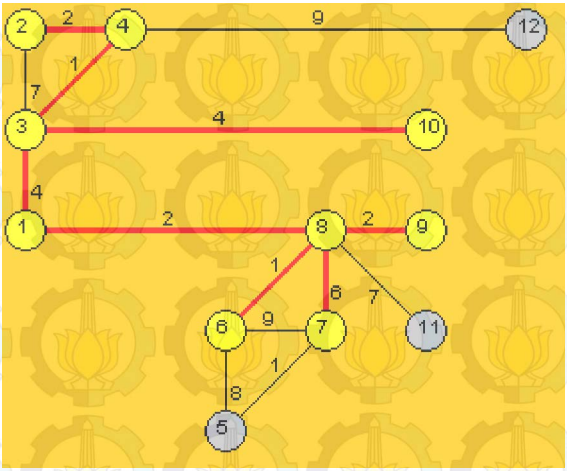
Pada *priority queue, top* saat ini adalah *edge(8,7)* dengan bobot 6. Setelah dicek *visited(7) = false*, maka set *visited(7) = true*, dan masukkan *edge(*8,7*)* ke dalam himpunan *mst. Dequeue top* dari *priority queue.* Visualisasi sampai tahap ini dapat dilihat pada Gambar 2.13. Masukkan semua *edge* yang berawal dari *vertex* 7 ke *vertex* yang belum pernah dikunjungi. *Priority queue* saat ini berisi {7-5:1, 3-2:7, 8-11:7, 6-5:8, 4-12:9, 6-7:9}.

Pada *priority queue, top* saat ini adalah *edge(7,5)* dengan bobot 1. Setelah dicek *visited(5) = false*, maka set *visited(5) = true*, dan masukkan *edge(*7,5*)* ke dalam himpunan *mst. Dequeue top* dari *priority queue.* Visualisasi sampai tahap ini dapat dilihat pada Gambar 2.14. Masukkan semua *edge* yang berawal dari *vertex* 5 ke *vertex* yang belum pernah dikunjungi. *Priority queue* saat ini berisi {3-2:7, 8-11:7, 6-5:8, 4-12:9, 6-7:9}.

Pada *priority queue, top* saat ini adalah *edge(3,2)* dengan bobot 7. Setelah dicek *visited(2) = true*. *Dequeue top* dari *priority queue. Priority queue* saat ini berisi {8-11:7, 6-5:8, 4-12:9, 6-7:9}.

Pada *priority queue, top* saat ini adalah *edge(8,11)* dengan bobot 7. Setelah dicek *visited(*11*) = false*, maka set *visited(*11*) = true*, dan masukkan *edge(*8,11*)* ke dalam himpunan *mst. Dequeue top* dari *priority queue.* Visualisasi sampai tahap ini dapat dilihat pada Gambar 2.15. Masukkan semua *edge* yang berawal dari *vertex* 11 ke *vertex* yang belum pernah dikunjungi. *Priority queue* saat ini berisi {6-5:8, 4-12:9, 6-7:9}.

Pada *priority queue, top* saat ini adalah *edge(6,5)* dengan bobot 8. Setelah dicek *visited(5) = true*. *Dequeue top* dari *priority queue. Priority queue* saat ini berisi {4-12:9, 6-7:9}.

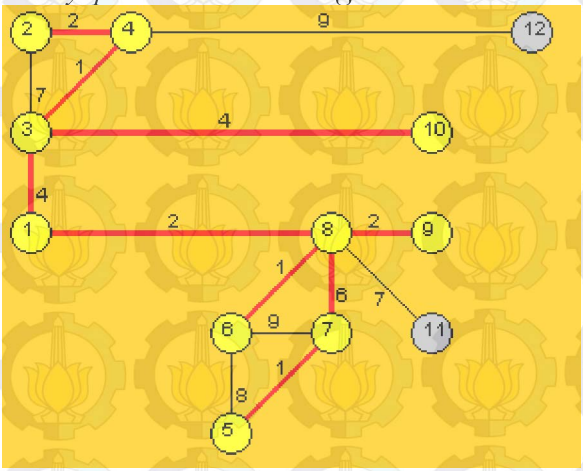


Gambar 2.13. Proses Algoritma Prim (9)

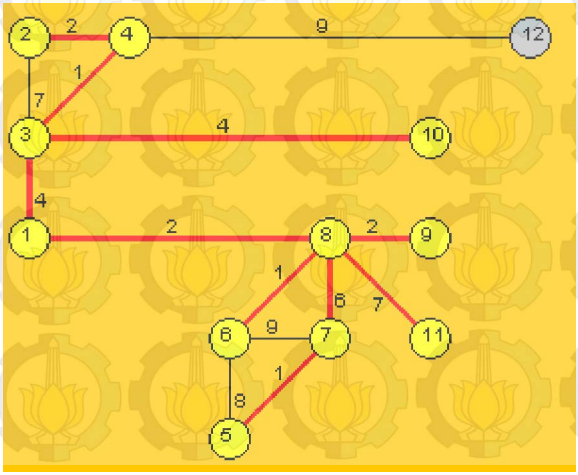
Pada *priority queue, top* saat ini adalah *edge(4,12)* dengan bobot 9. Setelah dicek *visited(*12*) = false*, maka set *visited(*12*) = true*, dan masukkan *edge(*6,5*)* ke dalam himpunan *mst. Dequeue top* dari *priority queue.* Visualisasi sampai tahap ini dapat dilihat pada Gambar 2.16. Masukkan semua *edge* yang berawal dari *vertex*

12 ke *vertex* yang belum pernah dikunjungi. *Priority queue* saat ini berisi {6-7:9}.

Pada *priority queue, top* saat ini adalah *edge(6,7)* dengan bobot 7. Setelah dicek *visited(9) = true*. *Dequeue top* dari *priority queue*. *Priority queue* saat ini berisi {}.

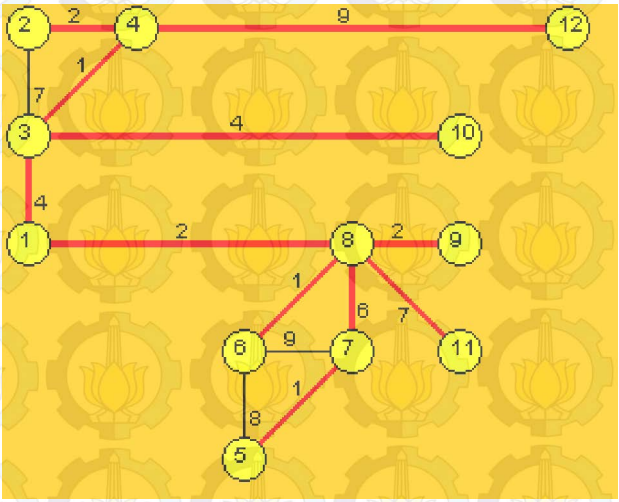


Gambar 2.14. Proses Algoritma Prim (10)



Gambar 2.15. Proses Algoritma Prim (11)

Jika *priority queue* kosong, maka *MST* telah ditemukan. Pada himpunan *mst*, *MST* terdiri dari *edge –edge : 1-8, 8-6, 8-9, 1-3, 3-4, 4-2, 8-7, 3-10, 7-5, 8-11, 4-12*. *Weight*, yang merupakan jumlah dari bobot *edge-edge* pada *MST* tersebut adalah 42.



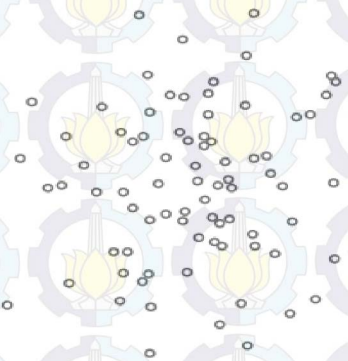
Gambar 2.16. Proses Algoritma Prim (12)

2.5 *Fast Minimum Spanning Tree*

Fast Minimum Spanning Tree (FMST) adalah metode untuk menyelesaikan permasalahan *MST* yang diusulkan oleh Caiming Zhong, dkk [6]. Metode ini tetap menggunakan algoritma konvensional, seperti algoritma Prim. Hanya saja, metode ini menerapkan konsep *Divide and Conquer*, dimana *graph* dibagi menjadi beberapa *subgraph*, dibentuk *MST* - nya masing-masing lalu dirangkai menjadi satu. Hasil yang diberikan oleh metode ini bukan merupakan hasil yang *unique*, tapi merupakan hasil perkiraan atau hasil yang mendekati. Akan tetapi, waktu yang diperlukan dalam metode ini untuk menyelesaikan permasalahan *MST* lebih sedikit dibanding algoritma Prim konvensional.

2.5.1 *Divide and Conquer*

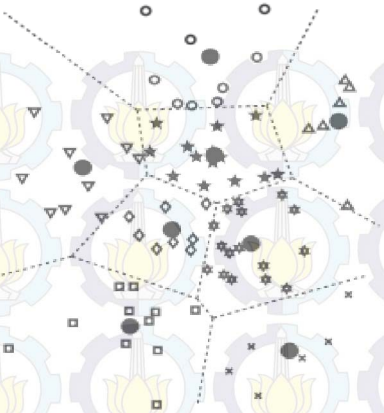
Pada tahap ini, *dataset* dibagi menjadi *k subset*, dimana *k* = √*N*. *Dataset* dibagi menggunakan K-Means yang menggunakan konsep kedekatan *locality*. Setelah data terbagi menjadi *k subset*, hitung jarak tiap data dalam *subset* dengan kombinasi pasangan data. Hal ini menyebabkan tiap *subset* seperti *complete graph* kecil, yang terdiri dari data didalamnya sebagai anggota dari himpunan *vertex,* dan *edge* yang menghubungkan tiap *vertex.* Lalu, algoritma *MST* konvensional diterapkan kepada tiap *subset*, sehingga akan terbentuk *k MST*. Sebagai contoh, diilustrasikan data berukuran 2 dimensi pada Gambar 2.17.



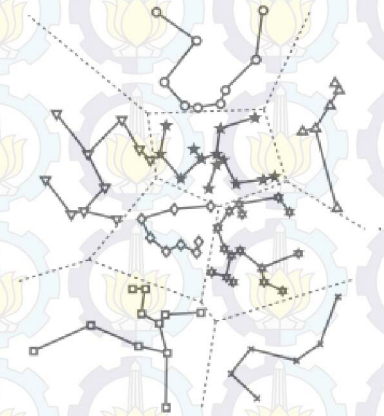
Gambar 2.17. Ilustrasi Dataset

Dataset pada Gambar 2.17 memiliki 78 data dibagi menjadi √78 kelompok, yaitu 8 kelompok menggunakan K-Means. Ilustrasi tersebut ditunjukkan pada Gambar 2.18. Pada Gambar 2.18, ada bulatan hitam yang merupakan *centroid /* titik pusat dari *subset.*

8 *subset* tersebut dicari *MST* nya dengan menggunakan algoritma konvensional *MST*. Sehingga terbentuk 8 *MST subset*. Ilustrasi tersebut dapat dilihat pada Gambar 2.19.



Gambar 2.18. Partisi Dataset ke beberapa subset



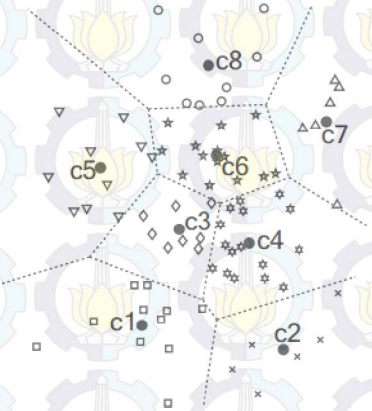
Gambar 2.19. Tiap subset dibentuk MST nya

2.5.2 *Combine Subset Algorithm*

Pada tahap ini, *k MST* yang terbentuk akan digabungkan. *Subset* yang digabungkan adalah *subset* yang bertetangga. *Subset* yang bertetangga dihubungkan dengan sebuah *edge* penghubung

yang ditentukan dengan cara di tahap selanjutnya, *Detecting the Connecting Edge. Subset* yang bertetangga merupakan *subset* yang *centroid*-nya dihubungkan dengan *MST centroid*.

Sebagai contoh, ditunjukkan pada Gambar 2.20, *dataset* terpartisi ke beberapa *subset*. Ditunjukkan bahwa setiap *subset* dicari *centroid*-nya dengan perwakilan bulatan hitam. *Centroid* merupakan titik tengah dari *subset.*



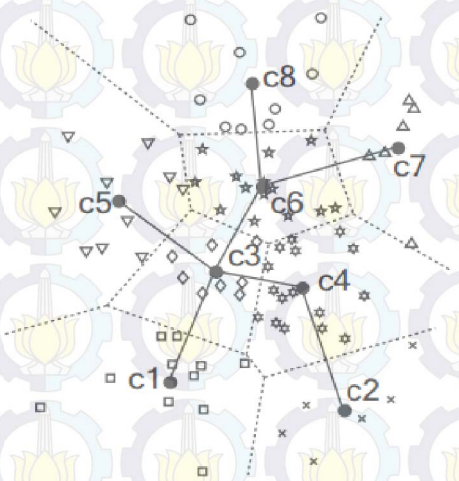
Gambar 2.20. Dataset yang terpartisi

Pada Gambar 2.21, suatu *MST* dibentuk dari *centroid* tiap *subset*. *Subset* yang *centroid* nya dihubungkan pada *MST centroid* merupakan *subset* yang bertetangga. *Subset c2* bertetangga dengan *subset c4*. *Subset c1* bertetangga dengan *subset c3.* Untuk setiap *subset* yang bertetangga dicari *edge* penghubungnya dengan tahap selanjutnya, *Detecting the Connecting Edge.* Hal ini ditunjukkan pada Gambar 2.22.

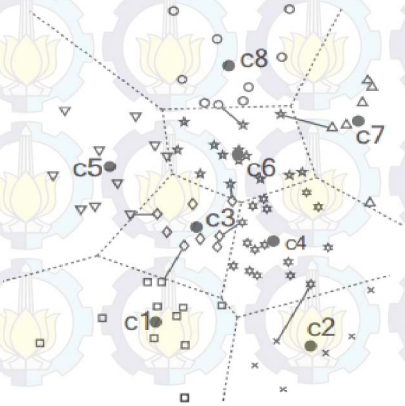
2.5.3 *Detecting the Connecting Edge*

Untuk setiap *subset* yang bertetangga, dicari *edge* yang menjadi penghubung antara dua *subset*. Cara yang digunakan adalah menghubungkan titik yang jaraknya paling dekat dengan *centroid subset* tetangga. Setelah semua *subset* terhubung, maka didapatkan satu *MST* perkiraan. Ilustrasi tahap ini ditunjukkan pada

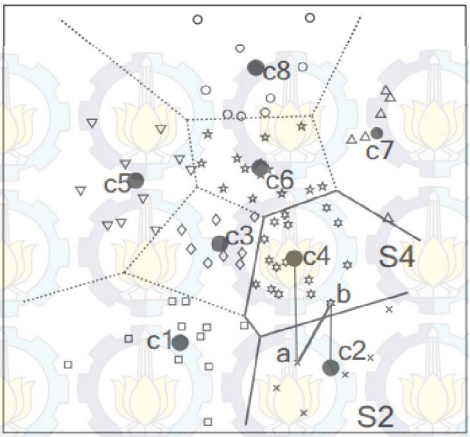
Gambar 2.23 [6]. Pada Gambar 2.23, ditunjukkan bahwa *vertex a* merupakan *vertex* didalam *subset S2* yang paling dekat dengan *centroid subset S4*. *Vertex b* merupakan *vertex* didalam *subset S4* yang paling dekat dengan *centroid subset S2*. Maka, *edge(a,b)* merupakan *edge* penghubung dari *subset S2* dan *S4*.



Gambar 2.21. MST centroid



Gambar 2.22. Subset yang bertetangga dicari edge penghubungnya

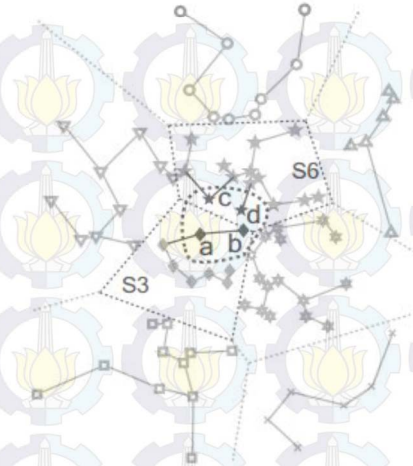


Gambar 2.23. Tahap Detecting the Connecting Edge

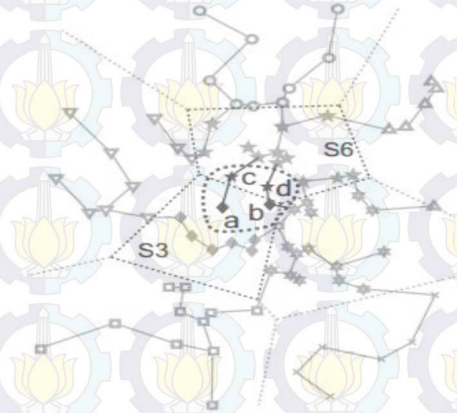
2.5.4 *Second Approximate MST*

Pada tahap ini, dilakukan pengulangan 3 tahap diatas, untuk memperbaiki hasil *MST* perkiraan pertama. Hasil *MST* perkiraan pertama sangat jauh bila dibandingkan dengan *MST* aslinya. Ini disebabkan pada saat *MST* per *subset* dibangun, data yang terletak pada perbatasan terbentuk di dalam *subset*, tidak melintasi *subset* yang lain. Pada Gambar 2.25 [6], dapat dilihat bahwa pada *MST* yang benar, *vertex* a menyambung dengan *vertex* c, dan *vertex* b menyambung dengan *vertex* d. Sedangkan pada *MST* perkiraan pertama yang ditunjukkan pada Gambar 2.24, *vertex* a menyambung dengan *vertex* b, *vertex* c menyambung dengan *vertex* d karena terdapat pada satu *subset*.

Data dibagi menjadi *k*-1 *subset* menggunakan algoritma klastering K-Means dengan cara menjadikan titik tengah dari *edge* - *edge* yang menjadi bagian dari *MST centroid* tiap *subset* menjadi *centroid* yang baru. Ilustrasi perhitunan *midpoint* ini ditunjukkan pada Gambar 2.26. Titik tengah ini biasanya terletak di sekitar perbatasan antar *subset*, sehingga dapat menyelesaikan permasalahan seperti yang ditunjukkan pada Gambar 2.24.

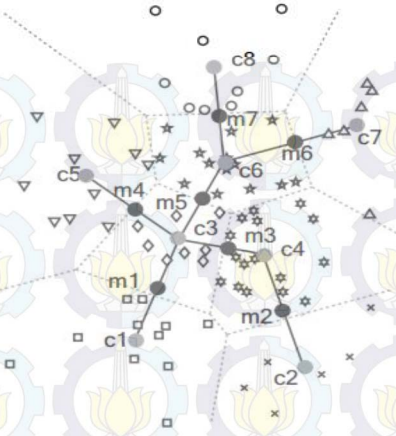


Gambar 2.24. Kasus Pembentukan MST yang Salah

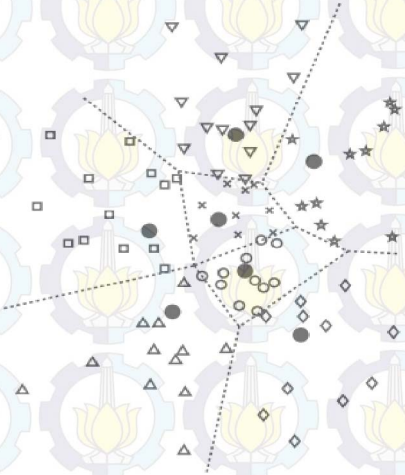


Gambar 2.25. Ilustrasi MST yang sebenarnya

Ilustrasi *dataset* dipartisi menggunakan *centroid* yang baru ditunjukkan pada Gambar 2.27. Lalu step *Combine Subset Algorithm* dan *Detecting the Connecting Edge* dilakukan. Pada akhir tahap ini, didapatkan *MST* perkiraan yang kedua.



Gambar 2.26. Perhitungan midpoint dari MSTcentroid



Gambar 2.27. Partisi menggunakan centroid baru

2.5.5 *Merge Algorithm*

Pada tahap ini, dua *MST* perkiraan yang didapat akan digabung menjadi sebuah *graph*. *Graph* ini memiliki paling banyak

memiliki *edge* sejumlah 2(*N-1*). Kemudian, *graph* ini dicari *MST* nya menggunakan algoritma *MST* konvensional. Pada akhir tahap ini, diperoleh *MST* perkiraan terakhir.

2.6 Klastering

Klastering adalah proses mengelompokkan data berdasarkan kemiripan antar data. Data yang memiliki kemiripan diletakkan pada satu kelompok yang sama, sedangkan yang tidak memiliki kemiripan diletakkan pada kelompok yang berbeda.

2.6.1 *K-Means*

K-Means adalah salah satu algoritma klastering (pengelompokan) yang paling umum. Algoritma ini mengelompokkan objek ke beberapa klaster, berdasarkan kedekatan dengan *centroid* [16]. Algoritma ini membutuhkan parameter jumlah klaster yang ingin dibentuk. *K-Means* dimulai dengan pemilihan *centroid* (titik tengah), satu untuk masing-masing klaster. Pemilihan *centroid* bisa secara acak, ataupun ditentukan. Setelah itu, masing-masing data yang ingin dikelompokkan dihitung jaraknya dengan masing-masing *centroid.* Perhitungan jarak dilakukan menggunakan *Euclidean Distance* dengan rumus (2.1).

Setelah dilakukan perhitungan jarak, data akan dimasukan ke dalam klaster yang sama berdasarkan *centroid* yang paling dekat. Hal ini terus dilakukan sampai data tidak berpindah-pindah klaster. Tahapan algoritma *K-Means* ditunjukkan pada Gambar 2.28. Sebagai contoh pada Gambar 2.29 [17]. Pada *step* 1, ditentukan 2 titik dari data sebagai *centroid.* Kemudian tiap data dikelompokkan dengan *centroid* yang terdekat. Pada *step 2* ditunjukkan bahwa *centroid* baru dihitung dengan cara menghitung rata-rata dari semua data yang terletak pada suatu kelompok. Pada *step 3* ditunjukkan, data dikelompokkan kembali dengan *centroid* baru yang didapat pada *step 2*. *Step* 4 mengilustrasikan penghitungan *centroid* yang baru untuk digunakan dalam

pengelompokan data pada *step 5*. Pada *step 6,* dilihat bahwa data pada tiap klaster tidak berubah. Maka, proses klastering selesai.

$$d(P,Q) = \sqrt{\sum_{j=1}^{n} (Xj(P) - Xj(Q))^2} \quad (2.1)$$

d(P,Q) = jarak / nilai Euclidean Distance titik P ke Q

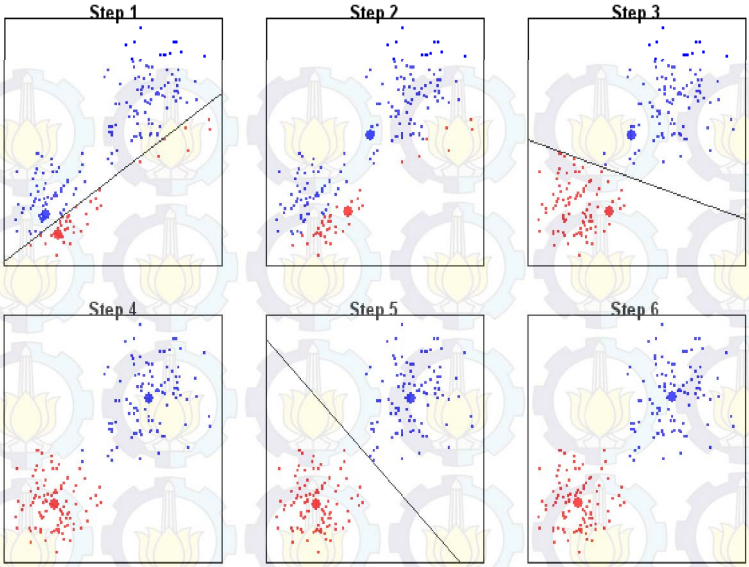
P = titik ke- 1 ; Q = titik ke-2

n = jumlah fitur

Xj (P) = nilai fitur ke j pada titik P

Masukan : *k*: Jumlah klaster
X: *Dataset* yang terdiri dari n objek
Keluaran : *k* klaster yang masing-masing berisi beberapa data
Metode :
(1). Tetapkan *k* titik dari *X* sebagai *centroid* (jika tidak ditentukan, maka pilih *k* titik secara *random*).
(2). Lakukan
Kelompokkan tiap data berdasarkan *centroid* yang paling dekat
Hitung *centroid* yang baru
(3). Sampai data dalam tiap klaster tidak berubah

Gambar 2.28. Tahapan Algoritma K-Means

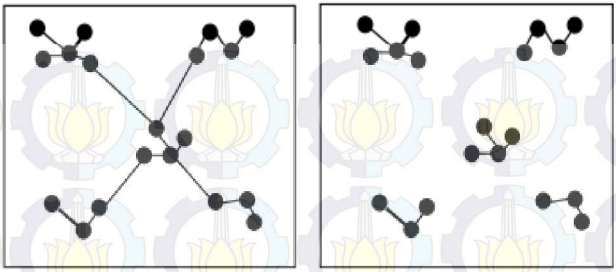


Gambar 2.29. Ilustrasi pembentukan klaster dengan K-Means

2.6.2 Klastering menggunakan *Minimum Spanning Tree*

Algoritma yang dipakai adalah algoritma yang diusulkan oleh Prasanta Jana dan Azad Naik. Algoritma ini ditujukan untuk melakukan pengelompokan data menggunakan informasi yang terdapat pada *MST*. Algoritma ini menggunakan konsep penghapusan *edge* yang melebihi *threshold*. *K edge* yang dihapus akan memisahkan *graph* menjadi *k+1 subgraph* yang terpisah dan dianggap merupakan klaster tersendiri.

Ilustrasi klastering menggunakan *MST* ditunjukkan pada Gambar 2.30 [7]. Pada gambar tersebut, dilakukan penghapusan 4 *edge* yang paling panjang, sehingga, terbentuk 5 klaster.



Gambar 2.30. Representasi MST dan klastering berdasarkan MST

2.7 Indeks Dunn

Indeks Dunn adalah suatu indeks yang dapat digunakan untuk menguji validitas klaster. Indeks ini merupakan rasio jarak terkecil antara dua titik yang terletak pada dua klaster yang berbeda dengan jarak terbesar antara dua titik yang terletak pada satu klaster [18]. Nilai indeks ini antara 0 sampai ∞. Semakin besar nilai indeks Dunn, maka hasil klastering dinyatakan bagus [19]. Rumus Indeks Dunn ditunjukkan pada rumus (2.2).

$$D = \min_{1\le i\le n}\{\min_{1\le j\le n, i\neq j}\{\frac{d(i,j)}{\max_{1\le k\le n} d'(k)}\}\} \quad (2.2)$$

D = nilai indeks Dunn
d(i,j) = jarak *Euclidean distance* antara titik pada klaster i dan klaster j
d'(k) = jarak *Euclidean distance* antara titik pada klaster k
n = jumlah klaster

2.8 Indeks Ray&Turi

Indeks Ray&Turi adalah suatu indeks yang dapat digunakan untuk menguji validitas klaster. Indeksi ini diusulkan oleh Ray dan Turi [7]. Konsep indeks ini berdasarkan *compactness* dan *isolation. Compactness* adalah ukuran kohesi antar data, dan *isolation* adalah ukuran perbedaan antar klaster.

Compactness diukur dengan rata-rata jarak semua titik dalam satu klaster ke *centroid* klaster tersebut. Rumus ini ditunjukkan pada rumus (2.3).

$$\text{Intra} = \frac{1}{\text{N}} \sum_{\text{i}=1}^{\text{k}} \sum_{x \in Ci} \|x - zi\|^2 \quad (2.3)$$

Intra = nilai *compactness* / intra-klaster
N= jumlah data
k = jumlah klaster
z_i = *centroid* klaster ke *i*
C_i = klaster ke *i*
x = data

Sedangkan *isolation*, diukur dengan mencari jarak antar dua *centroid* klaster yang paling kecil. Rumus ini ditunjukkan pada rumus (2.4).

$$Inter = \min \left\| z_i - z_j \right\|^2 ; i \neq j \quad (2.4)$$

Inter = nilai *isolation* / inter-klaster
z_i = *centroid* klaster ke *i*
z_j = *centroid* klaster ke *j*

Rasio yang merupakan hasil dari *compactness* dibagi *isolation*, merupakan nilai yang digunakan untuk menguji validitas klaster. Semakin kecil nilai indeks ini, maka hasil klastering dinyatakan semakin baik. Rumus ini ditunjukkan pada rumus (2.5).

$$Ratio(Validity) = \frac{Intra}{Inter} \quad (2.5)$$

Ratio= nilai validitas indeks klaster
Intra = nilai *compactness* / intra-klaster
Inter = nilai *isolation* / inter-klaster

BAB III
DESAIN PERANGKAT LUNAK

Pada bab ini akan dijelaskan perancangan program yang dibuat. Perancangan akan dibagi menjadi dua proses utama, yaitu:

1. Desain *Fast Minimum Spanning Tree*.
2. Desain klastering berdasarkan *Minimum Spanning Tree.*

Pada bab ini akan dijelaskan gambaran umum setiap program utama dalam *pseudocode*.

3.1 Desain Metode Secara Umum

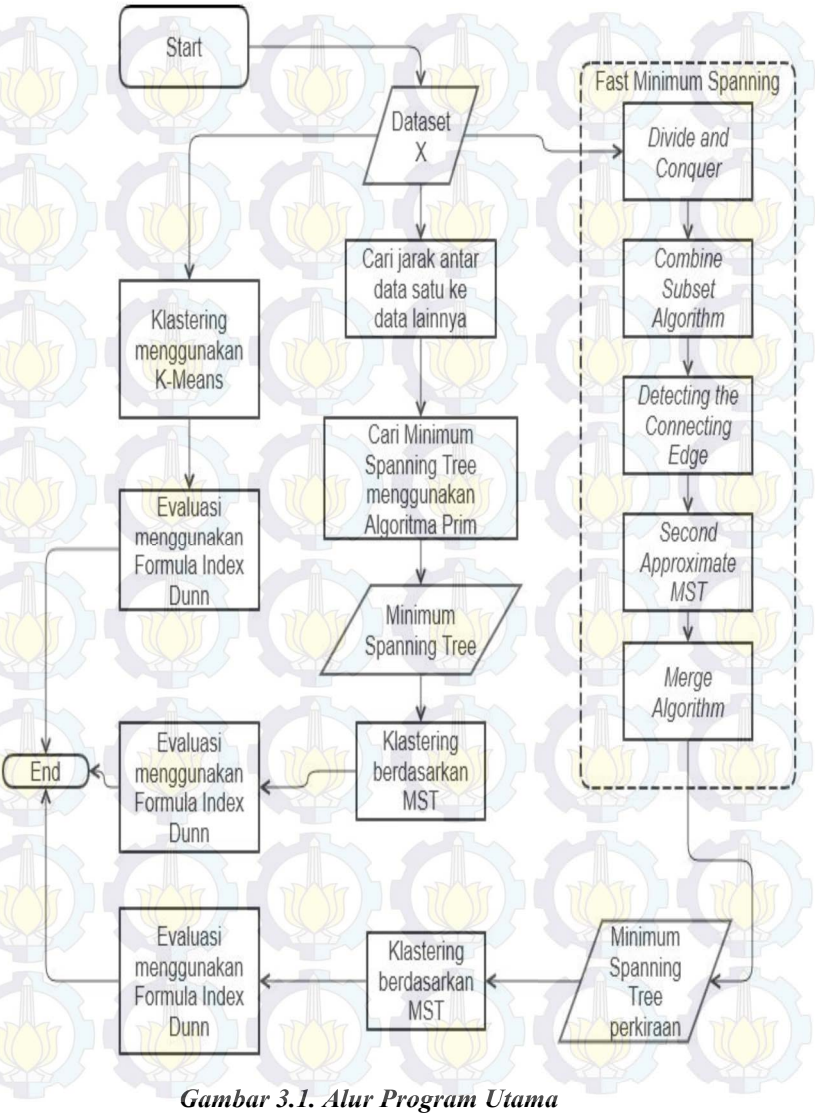
Pada tugas akhir ini akan dibangun suatu sistem yang dapat mengimplementasikan *Fast Minimum Spanning Tree (FMST)* dan klastering menggunakan *Minimum Spanning Tree (MST)*. Sistem akan menerima masukan *dataset* yang mengandung data bersifat numerik.

Dalam tugas akhir ini, proses pertama yang dilakukan adalah melakukan metode *FMST* terhadap *dataset* masukan. Kemudian, mencari jarak tiap satu data dengan data lainnya dengan rumus *Euclidean* distance. Kemudian, *dataset* tersebut dicari *MST* nya menggunakan metode algoritma Prim.

Klastering dilakukan menggunakan *MST* dari metode *FMST* dan algoritma Prim, serta menggunakan K-Means. Kemudian hasil klastering dari 3 metode tersebut diuji menggunakan rumus Indeks Dunn. Alur Program Utama terdapat pada Gambar 3.1.

3.2 Desain Algoritma Prim

Algoritma ini digunakan untuk menyelesaikan permasalahan *MST*. Algoritma Prim yang digunakan menggunakan struktur data *priority queue. Priority queue* tersebut didesain berdasarkan *low priority,* dimana data yang bernilai lebih kecil diprioritaskan. Desain Algoritma Prim terdapat pada Gambar 3.2.



Gambar 3.1. Alur Program Utama

Masukan	*Graph G = (V,E)*, himpunan *vertex V* dan himpunan *edge E*
Keluaran	*Minimum Spanning Tree* dari *G*

```
1. Tentukan s sebagai vertex awal; s ∈ V
2. Set visited(s) = true
3. Masukkan semua edge yang memiliki vertex awal s
   beserta bobot edge tersebut kedalam priority
   queue
4. Selama priority queue tidak kosong,ulangi :
   Ambil puncak dari priority queue =>edge(s, x),w;
   s merupakan vertex awal, x merupakan vertex
   akhir, dan w merupakan bobot dari edge yang
   menghubungkan vertex s dan vertex x.
   Masukkan edge tersebut sebagai anggota himpunan
   dari MST.
   Hapus puncak dari priority queue.
   Jika visited(x) != true, lakukan langkah ke 5
5. Set visited(x) = true
   Masukkan semua edge yang memiliki vertex awal x
   dan visited(y)!= true, dimana y adalah vertex
   akhir, beserta bobot edge tersebut kedalam
   priority queue
6. Selesai
```

Gambar 3.2. Pseudocode Algoritma Prim

3.3 Desain Metode *Fast Minimum Spanning Tree*

Pada bagian ini dijelaskan tiap tahapan dari *Fast Minimum Spanning Tree* disertai *pseudocode*. *FMST* terdiri dari 5 tahap, *Divide and Conquer, Combine Subset Algorithm, Detecting the Connecting Edge, Second Approximate MST*, dan *Merge Algorithm*. Desain *FMST* secara keseluruhan dapat dilihat pada Gambar 3.3.

Masukan	*Dataset X* yang terdiri dari *N* data dan *m* fitur.
Keluaran	*Minimum Spanning Tree* perkiraan dari *dataset X*

1. Terapkan *Divide and Conquer* terhadap *X* untuk mendapatkan *k* buah *MST*
2. Terapkan *Combine Subset Algorithm* untuk mencari *subset* yang bertetangga dan *Detecting the Connecting Edge* untuk mendapatkan *edge* yang menggabungkan *subset* yang bertetangga. Tahap ini menghasilkan *MST* perkiraan pertama.
3. Terapkan Second Approximate *MST* untuk mendapatkan *MST* perkiraan kedua.
4. Gabung *MST* perkiraan pertama dan kedua untuk mendapatkan *graph G*.
5. Terapkan algoritma Prim pada *G* untuk mendapatkan *MST* perkiraan dari *X*.

Gambar 3.3. Pseudocode Metode Fast Minimum Spanning Tree

3.3.1 Tahap *Divide and Conquer*

Tahap *Divide and Conquer* adalah tahap pertama dalam metode *FMST*. *Dataset* dipartisi ke $\sqrt{N}$ *subset*, dimana *N* merupakan jumlah data dengan menggunakan algoritma klastering K-Means. Pada tiap *subset*, hitung jarak antar tiap data didalamnya dengan menggunakan *Euclidean distance*. Perhitungan dilakukan untuk setiap kombinasi pasangan data. Langkah ini menjadikan tiap *subset* seperti menjadi *complete graph* tersendiri dengan tiap data didalam *subset* menjadi *vertex* dan jarak antar tiap data menjadi bobot dari *edge* yang menghubungkan pasangan data. Tiap *subset* dicari *MST* nya dengan menggunakan algoritma Prim.

Masukan pada tahap ini adalah *dataset* yang ingin dicari *MST* nya. Keluaran pada tahap ini adalah $\sqrt{N}$ *MST* dari *subset*. Desain tahap *Divide and Conquer* terdapat pada Gambar 3.4.

Masukan	*Dataset X*, yang terdiri dari *N* data dan m fitur.
Keluaran	*k* buah *MST subset* => MST_i ; $1 \leq i \leq k$

1. Set k= $\sqrt{N}$
2. Terapkan algoritma klastering Kmeans dengan parameter *k* ke *dataset X* untuk mendapatkan sebanyak *k subset*. Simpan pada variabel $S=\{S_1,S_2,\ldots,S_k\}$
3. Untuk setiap *subset*, hitung jarak antar satu data dengan data lainnya dalam *subset* tersebut menggunakan *Euclidean Distance* untuk menghasilkan *k complete graph* $P=\{P_1,P_2,\ldots,P_k\}$; $1 \leq i \leq k$
4. Terapkan algoritma Prim pada setiap anggota dari *P* untuk mendapatkan *k MST* => MST_i ; $1 \leq i \leq k$

Gambar 3.4. Pseudocode Tahap Divide and Conquer

3.3.2 Tahap *Combine Subset Algorithm*

Tahap *Combine Subset Algorithm* adalah tahap untuk menggabungkan *k MST* yang telah dihasilkan pada tahap *Divide and Conquer*. Untuk menggabungkan *k MST*, diperlukan *edge* yang menghubungkan *subset* yang bertetangga. *Subset* yang bertetangga adalah *subset* yang *centroid* (titik pusat) nya dihubungkan pada *MSTcentroid*. *MSTcentroid* adalah *MST* dari *graph* yang terdiri dari *centroid* tiap *subset* sebagai *vertex*, serta *edge* yang menghubungkan tiap *centroid* tersebut. *Edge* penghubung *subset* yang bertetangga diselesaikan dengan cara pada tahap selanjutnya yaitu, *Detecting the Connecting Edge*. Setelah didapat *k-1 edge* penghubung, tambahkan *k-1 edge* tersebut dengan *k MST* untuk membentuk suatu himpunan yang merupakan *MST* perkiraan pertama. Desain tahap *Combine Subset Algorithm* dapat dilihat pada Gambar 3.5.

Masukan	*k* buah *MST subset* => MST_i ; $1 \leq i \leq k$
Keluaran	*MST* perkiraan pertama dari *dataset X* => *MST1* *MST* dari tiap *centroid subset* => *MSTcen*

1. Hitung *centroid subset* S_i => C_i; $1 \leq i \leq k$. *Centroid* merupakan titik tengah/ pusat dari data pada *subset* tersebut.
2. Hitung jarak tiap C_i *menggunakan Euclidean Distance* untuk membentuk *graph G*, dimana semua C_i menjadi anggota himpunan *vertex V* dan jarak tiap C_i menjadi anggota himpunan *edge E*.
3. Terapkan algoritma Prim pada *graph G* untuk mendapatkan *MSTcen*.
4. Untuk setiap *subset* yang *centroid* nya dihubungkan oleh *edge e*, dimana $e \in$ *MSTcen*, terapkan *Detecting Connecting Edge* untuk mendapatkan *edge* penghubung.
5. Gabung *k-1 edge* penghubung dengan *k MST* menjadi *MST1*.

Gambar 3.5. Pseudocode Combine Subset Algorithm

3.3.3 Tahap *Detecting the Connecting Edge*

Pada tahap ini, setiap pasangan *subset* yang dijadikan sebagai masukan, akan dicari *edge* yang menghubungkannya. *Edge* tersebut terdiri dari *vertex* dalam *subset* yang paling dekat dengan *centroid subset* pasangannya. Konsep jarak yang dipakai adalah *Euclidean distance.* Jika terdapat *k subset*, maka terdapat *k*-1 *edge* penghubung. Desain tahap *Detecting the Connecting Edge* terdapat pada Gambar 3.6.

3.3.4 Tahap *Second Approximate MST*

Tahap ini merupakan tahap untuk mencari *MST* perkiraan yang kedua. Cara yang digunakan sama dengan cara yang digunakan untuk mendapatkan *MST* perkiraan yang pertama.

Hanya saja, partisi yang digunakan berbeda. *MSTcen* yang didapat pada tahap *Connecting Subset Algorithm* dihitung titik tengahnya (*midpoint*). Tiap data dikelompokkan menggunakan algoritma K-Means dengan menginisialisasi *midpoint* sebagai *centroid* untuk membentuk *k-1 subset* yang baru. Lalu tiap *subset* tersebut dicari *MST* nya masing-masing dengan algoritma Prim. Kemudian, terapkan tahap *Combine Subset Algorithm* pada *k-1 subset* tersebut. Pada akhir tahap ini, didapatkan *MST* perkiraan yang kedua. Desain tahap *Second Approximate MST* terdapat pada Gambar 3.7.

Masukan	Pasangan *subset* yang akan dihubungkan =>(S_i, S_j)
Keluaran	*Edge e* yang menghubungkan *subset* S_i dan *subset* S_j
1. Cari *a*, *vertex* pada *subset* S_i yang paling dekat dengan *centroid subset* S_j. 2. Cari *b*, *vertex* pada *subset Sj* yang paling dekat dengan *centroid subset* S_i. 3. *Edge e*, *edge* yang menghubungkan *vertex a*, dan *vertex b*, merupakan penghubung *subset* S_i dan S_j.	

Gambar 3.6. Pseudocode Detecting the Connecting Edge

Masukan	*MSTcen*, *Dataset X* yang terdiri dari N data dan m fitur
Keluaran	*MST* perkiraan kedua dari *dataset X* => *MST*2
1. Hitung *midpoint* (titik tengah) untuk setiap *edge e* ∈ *MSTcen*. 2. Partisi *dataset X* dengan algoritma klastering K-Means menggunakan *midpoint* sebagai *centroid* menjadi *k-1 subset* $S'=\{S'_1, S'_2, \ldots, S'_{k-1}\}$. 3. Hitung jarak antar satu data dengan data lainnya menggunakan *Euclidean Distance* dalam *subset* S'i untuk menghasilkan *k-1* complete *graph* $P'=\{P'_1,P'_2,\ldots,P'_{k-1}\}$; $1 \leq i \leq k-1$. 4. Terapkan algoritma Prim pada tiap anggota *P'* untuk mendapatkan *k-1 MST'* => MST'_i; $1 \leq i \leq k-1$. 5. Terapkan *Combine Subset Algorithm* pada *k-1 MST'* untuk mendapatkan *MST* perkiraan kedua.	

Gambar 3.7. Pseudocode Second Approximate MST

3.3.5 Tahap *Merge Algorithm*

Tahap ini adalah tahap paling akhir dalam metode *FMST*. *MST1* dan *MST2* yang merupakan *MST* perkiraan yang didapat pada tahap-tahap sebelumnya digabung menjadi sebuah *graph*. *Graph* ini memiliki tidak lebih dari 2(*N*-1) *edge.* Kemudian, algoritma Prim diterapkan pada *graph* tersebut untuk mendapatkan *MST* perkiraan terakhir. Desain tahap *Merge Algorithm* terdapat pada Gambar 3.8.Gambar 3.7

Masukan	*MST1, MST2*
Keluaran	*MST* perkiraan terakhir => *FMST*
1. Gabung *MST1* dan *MST2* menjadi sebuah *graph G*. 2. Terapkan algoritma Prim terhadap *G* untuk mendapatkan *MST* perkiraan terakhir => *FMST*.	

Gambar 3.8. Pseudocode Merge Algorithm

3.4 Desain Metode Klastering berdasarkan *Minimum Spanning Tree*

Metode ini adalah metode yang diusulkan oleh Prasanta Jana dan Azad Naik. Konsep pengelompokan dari metode ini adalah dengan cara menghapus *edge* yang ada pada *MST dataset*. Dengan menghapus *k edge,* maka akan menghasilkan *k+1* klaster. *Edge* yang dihapus merupakan *edge* yang terpanjang atau memiliki bobot melebihi *threshold* yang telah ditentukan. Hasil klastering yang diperoleh, dihitung rasio nya dengan rumus yang diusulkan Ray & Turi. Ulangi langkah-langkah diatas dengan menambah nilai *threshold.* Langkah- langkah tersebut diulangi sampai semua *edge* telah terhapus. Indikator klastering tersebut adalah yang paling baik, adalah ketika rasio yang didapatkan bernilai paling kecil. Desain Metode klastering berdasarkan *MST* terdapat pada Gambar 3.9.

Masukan	*Minimum Spanning Tree*, himpunan *edge* yang menghubungkan *vertex awal dan vertex akhir* dengan bobot *Euclidean Distance*, dari *dataset X*
Keluaran	*threshold*
1. Tentukan nilai *threshold*, dan *step_size*. 2. Tentukan nilai perulangan *i*=1. 3. Selama *threshold* < *edge* terpanjang dari *Minimum Spanning Tree*, ulangi : Hapus *edge* yang memiliki bobot lebih besar dari *threshold*. Untuk setiap *k edge* yang dihapus, maka tercipta *k-1* klaster. Evaluasi hasil klastering menggunakan uji validitas index Ray&Turi => *ratio(i)*. *threshold = threshold + step_size.* *i=i+1.* 4. Mengembalikan nilai threshold pada perulangan ke *i*, dimana *ratio(i)* merupakan nilai terkecil dibanding *ratio* lainnya.	

Gambar 3.9. Pseudocode klastering menggunakan MST

3.5 Desain Metode Indeks Dunn

Indeks Dunn merupakan suatu indeks yang dapat digunakan untuk menguji validitas klaster. Desain metode ini dapat dilihat pada. Desain metode Indeks Dunn dapat dilihat pada Gambar 3.10.

Masukan	*Dataset X*, yang terdiri dari m data dan n+1 fitur. Data uji coba memiliki n=2 fitur atau dimensi. Fitur terakhir berisi label klaster setiap data
Keluaran	Nilai uji evaluasi Indeks Dunn
1. Cari nilai *a* yang merupakan jarak terpendek antar data berdasarkan *Euclidean Distance* dalam klaster berbeda. Pencarian dilakukan pada semua kombinasi pasangan data. 2. Cari nilai *b* yang merupakan jarak terpanjang antar data berdasarkan *Euclidean Distance* dalam klaster yang sama. Pencarian dilakukan pada semua kombinasi pasangan data. 3. Mengembalikan nilai uji evaluasi = *a/b* sebagai keluaran.	

Gambar 3.10. Pseudocode Metode Indeks Dunn

[Halaman ini sengaja dikosongkan]

BAB IV
IMPLEMENTASI

Pada bab ini akan dibahas mengenai implementasi yang dilakukan berdasarkan rancangan yang telah dijabarkan pada bab sebelumnya. Sebelum penjelasan implementasi akan ditunjukkan terlebih dahulu lingkungan untuk melakukan implementasi.

4.1 Lingkungan Implementasi

Lingkungan implementasi yang akan digunakan untuk melakukan implementasi adalah MATLAB yang diinstal pada sistem operasi *Windows* 8.1.

4.2 Implementasi

Pada subbab ini akan dijelaskan implementasi setiap subbab yang terdapat pada bab sebelumnya yaitu bab perancangan program. Pada bagian implementasi ini juga akan dijelaskan mengenai fungsi-fungsi yang digunakan dalam program tugas akhir ini dan disertai dengan kode sumber masing-masing fungsi utama.

4.2.1 Implementasi Tahap *Divide and Conquer*

Tahap *Divide and Conquer* adalah tahap pertama pada *Fast Minimum Spanning Tree (FMST)*, dimana *dataset* dibagi menjadi beberapa *subset* lalu diselesaikan *MST* nya per *subset*. *Dataset* dibagi menggunakan metode K-Means. Metode K-Means dapat dipanggil secara langsung karena disediakan sebagai *library* oleh MATLAB. *Dataset* dibagi menjadi √N *subset* dengan menggunakan *Euclidean distance*, dimana *centroid* merupakan rata-rata semua titik dalam satu *subset*.

Implementasi Tahap *Divide and Conquer* dapat dilihat pada Kode Sumber 4.1. Fungsi tersebut diimplementasikan pada

baris 1 dan baris ke 2. Pada baris ke 3 hingga baris ke 19 dijelaskan untuk setiap *subset*, dilakukan implementasi algoritma Prim untuk menyelesaikan permasalahan *MST* nya. Lalu hasil *MST* dari tiap *subset* ditampung di variable *mst1*.

```
1  Cluster  =floor(sqrt(size(X,1)));
2   [Index, Centroid] =
   kmeans(Data,Cluster,'distance','sqEuclidean,'
   Replicates',Cluster);
3  for i=1:size(Centroid,1)
4      temp=[]; count=1;
5      for j=1:size(Data,1)
6          if Index(j)== i
7              temp(count,:)= Data(j,:);
8              count=count+1;
9          end
10     end
11     if(size(temp,1)>1)
12         [mst,cost] = prim(temp);
13         plus = repmat(counter-1,size(mst));
14         mst1 = [mst1;mst+plus];
15     end
16     temp(:,end+1)= i;
17     sorted(counter:counter+count-2,:)        =
   temp();
18     counter=counter+count-1;
19 end
```

Kode Sumber 4.1. Kode Sumber Tahap Divide and Conquer

4.2.2 Implementasi Tahap *Combine Subset Algorithm*

Tahap *Combine Subset Algorithm* adalah tahap dimana *MST* dari tiap *subset* digabungkan. *Subset* yang digabungkan merupakan *subset* yang bertetangga. *Subset* yang bertetangga adalah *subset* yang *centroid* nya dihubungkan oleh *MSTcentroid*.

Implementasi Tahap *Combine Subset Algorithm* dapat dilihat pada Kode Sumber 4.2.

```
1  mstCentroid,costCentroid] = prim(Centroid);
```

Kode Sumber 4.2. Kode Sumber Tahap Combine Subset Algorithm

4.2.3 Implementasi Tahap *Detecting the Connecting Edge*

Pada tahap ini, dicari *edge* yang menghubungkan tiap *subset* yang bertetangga. Implementasi tahap *Detecting the Connecting Edge* dapat dilihat pada kode sumber Kode Sumber 4.3. *Edge* tersebut dibentuk oleh titik yang memiliki jarak paling kecil dengan *centroid* dari *subset* tetangga nya. Langkah ini dilakukan pada baris ke 2 hingga ke 17. Selanjutnya *edge* - *edge* tersebut digabungkan dengan *mst1* sebagai himpunan dari *MST* perkiraan pertama. . Langkah ini dilakukan pada baris ke 18.

```
1  edge=[];
2  for i=1:size(mstCentroid,1)
3      for j=1:size(mstCentroid,2)
4          minimum = Inf;
5          for k=1:size(sorted,1)
6              if sorted(k,end) ==
   mstCentroid(i,j)
7                  x =
   norm(Centroid(mstCentroid(i,end-j+1),:) -
   sorted(k,1:end-1));
8                  if x < minimum
9                      minimum = x;
10                     edge(i,j) = k;
11                 end
12             elseif sorted(k,end) >
   mstCentroid(i,j)
13                 break;
14             end
15         end
16     end
17 end
18 mst1 = [mst1;edge];
```

Kode Sumber 4.3. Kode Sumber Tahap Detecting the Connecting Edge

4.2.4 Implementasi Tahap *Second Approximate MST*

Tahap ini digunakan untuk mendapatkan *MST* perkiraan kedua. Tahap ini dilakukan sebagai tahap untuk memperbaiki *MST* perkiraan pertama yang hasilnya jauh dari *MST* aslinya akibat data yang terletak pada perbatasan tidak terselesaikan dengan baik.

Implementasi tahap ini hampir sama dengan implementasi untuk mendapatkan *MST* perkiraan pertama, dimana membutuhkan tahap *Divide and Conquer, Combine Subset Algorithm,* dan *Detecting the Connecting Edge.* Perbedaannya terletak pada partisi *dataset*. *Dataset* yang terdiri dari *N* data dibagi menjadi *√N-1 subset* berdasarkan kedekatan titik dengan *centroid* yang merupakan titik tengah dari *edge-edge* pada *MSTcentroid*. Implementasi untuk perhitungan *centroid* yang baru dapat dilihat pada Kode Sumber 4.4. Kode Sumber 4.5, dan Kode Sumber 4.6 merupakan implementasi *Second Approximate MST* yang memuat 3 tahap diatasnya, *Divide and Conquer* (baris 3-20), *Combine Subset Algorithm* (baris 31), dan *Detecting the Connecting Edge* (baris 32-55).

```
1  newCentroid = [];
2  for i=1:size(mstCentroid,1)
3      newCentroid(i,:) =
   (Centroid(mstCentroid(i,1),:) +
   Centroid(mstCentroid(i,2),:))/2;
4  end
5  sorted(:,end+1)=0;
6  for i=1:size(sorted,1)
7      minimum = Inf;
8      for j=1:size(newCentroid,1)
9          x = norm(sorted(i,1:size(Data,2))-
   newCentroid(j,:));
10         if x<minimum
11             minimum=x;
12             sorted(i,end) = j;
13         end
14     end
15 end
```

Kode Sumber 4.4. Kode Sumber Tahap Second Approximate MST

```
1  count=1;counter=1;newsorted=[];mst2=[];no=[];
2  %Build mst per subset
3  for i=1:size(newCentroid,1)
4      temp=[];
5      count=1;
6      for j=1:size(sorted,1)
7          if sorted(j,end)== i
8              temp(count,:)=
   sorted(j,1:size(Data,2));
9              no = [no;j];
10             count=count+1;
11         end
12     if(size(temp,1)>1)
13         [mst,cost] = prim(temp);
14         plus = repmat(counter-1,size(mst));
15         mst2=[mst2;mst+plus];
16     end
17     temp(:,end+1)= i;
18     newsorted(counter:counter+count-2,:) =
   temp();
19     counter=counter+count-1;
20 end
21 %hitung new centroid
22 newCentroidx=[];
23 newnoCluster=[];
24 for i=1:size(newCentroid,1)
25     if size(find(newsorted(:,end)==i),1)>0
26         newCentroidx=
   [newCentroidx;newCentroid(i,:)];
27         newnoCluster = [newnoCluster;i];
28     end
29 end
30 %Combine Subset
31 [mstnewCentroid,costnewCentroid] =
   prim(newCentroidx);
32 %Detecting the Connecting Edge
33 ansx=zeros(1,size(mstnewCentroid,1)*
   size(mstnewCentroid,2));
34 for i =1 :
   size(mstnewCentroid,1)*size(mstnewCentroid,2)
```

Kode Sumber 4.5. Kode Sumber Tahap Second Approximate MST (2)

```
35      ansx(i) = newnoCluster(mstnewCentroid(i));
36  end
37  mstnewCentroid =
    reshape(ansx,[size(ansx,2)/2 , 2]);
38  edge=[];
39  for i=1:size(mstnewCentroid,1)
40      for j=1:size(mstnewCentroid,2)
41          minimum = Inf;
42          for k=1:size(newsorted,1)
43              if newsorted(k,end) ==
    mstnewCentroid(i,j)
44                  x =
    norm(newCentroid(mstnewCentroid(i,end-j+1),:)
    - newsorted(k,1:end-1));
45                  if x < minimum
46                      minimum = x;
47                      edge(i,j) = k;
48                  end
49              elseif newsorted(k,end) >
    mstnewCentroid(i,j)
50                      break;
51              end
52          end
53      end
54  end
55  mst2 = [mst2;edge];
56  % hold on
57
58  mst2adjust=[];
59  for i =1 : (size(mst2,1)*size(mst2,2))
60      mst2adjust(i)= no(mst2(i));
61  end
62  mst2adjust =
    reshape(mst2adjust,[size(mst2adjust,2)/2 ,
    2]);
63
64  mst2 = sort(mst2adjust')';
65  mst1= sort(mst1')';
66
```

Kode Sumber 4.6. Kode Sumber Tahap Second Approximate MST (3)

4.2.5 Implementasi Tahap *Merge Algorithm*

Implementasi tahap ini ditunjukkan pada Kode Sumber 4.7. Pada tahap ini, *MST* perkiraan pertama dan kedua digabung menjadi satu *graph* (pada baris 2-3). Lalu algoritma Prim diterapkan pada *graph* tersebut untuk mendapatkan *MST* perkiraan terakhir.

```
1  G = [mst1;mst2];
2  G = unique(G,'rows');
3  x = unique(G(:,1:2));
4  [mstfinal costfinal] =
   primadjlistedit(G,size(Data,1));
```

Kode Sumber 4.7. Kode Sumber Tahap Merge Algorithm

4.2.6 Implementasi Algoritma Prim

Algortima Prim digunakan untuk menyelesaikan permasalah *MST*. Untuk implementasi ini membutuhkan struktur data *adjacency list* dan *min weight priority queue*. List digunakan untuk menyimpan informasi titik tetangga dan *weight*. Sedangkan *priority queue* diimplementasikan berdasarkan *weight* yang paling bernilai minimum. Implementasi ini dapat dilihat pada Kode Sumber 4.8 dan Kode Sumber 4.9.

```
1  function [mst , jumlah] = prim(data)
2  PV = pdist(data,'euclidean');
3  a=list();
4  visited=zeros(1,size(data,1));
5  for i=1:size(data,1)
6      a.insert(i,list());
7  end
8  counter=1;
9  for i = 1 : size(data,1)-1
10     for j = i+1 : size(data,1)
11         weight = PV(counter);
12         a.getValue(i).insert(j,weight);
13         a.getValue(j).insert(i,weight);
14         counter = counter+1;
```

Kode Sumber 4.8. Kode Sumber Algoritma Prim

```
15      end
16  end
17  % tic
18  start = a.peek();
19  pqx=[];
20  for i=1:a.getValue(start).size
21      [c d] = a.getValue(start).get(i);
22      pqx = [pqx;d a.get(start) c];
23  end
24  pqx = sortrows(pqx,1);
25  visited(1,start)=1;
26  mst=zeros(size(data,1)-1,3);
27  counter=1;
28
29  while(size(pqx,1)>0)
30      s = pqx(1,1);
31      t = pqx(1,2);
32      u = pqx(1,3);
33      pqx = pqx(2:end,:);
34      if visited(1,u) ==0
35          visited(1,u) =1;
36          mst(counter,:) = [t,u,s];
37          counter = counter+1;
38          for i=1:a.getValue(u).size
39              [c d] = a.getValue(u).get(i);
40              if visited(1,c)==0
41                  pqx = [pqx;d a.get(u) c];
42              end
43          end
44          pqx = sortrows(pqx,1);
45      end
46  end
47
48  jumlah = sum(mst(:,3));
49
50  mst = mst(:,1:2);
```

Kode Sumber 4.9. Kode Sumber Algoritma Prim (2)

4.2.7 Implementasi Klastering berdasarkan *Minimum Spanning Tree*

Klastering berdasarkan *MST* menggunakan cara penghapusan *edge* pada *MST* yang memiliki bobot melebihi *threshold*. Setelah itu dihitung rasio antara *intra* klaster dan *inter* klaster. Hal tersebut diulangi terus menerus dengan memperbarui *threshold* sampai tidak ada lagi *edge* yang dihapus. *Threshold* diisi dengan nilai persentase, karena tiap *dataset* memiliki nilai rentang yang berbeda. Implementasi metode ini ditunjukkan pada Kode Sumber 4.10, Kode Sumber 4.11, Kode Sumber 4.12, dan Kode Sumber 4.13.

```
1   function[C thresholdx l ratio] =
    main_driver(mst,data,threshold,step_size)
2   [N,d]=size(data); % N,d holds the dimension of
    data
3   Inter=0.0;Intra=0.0;count=1;it=1;T=zeros(N,N)
    ;signal=0;
4   sorted=data;
5   for i=1:size(mst,1)
6           A(mst(i,1),mst(i,2))=1;
7           A(mst(i,2),mst(i,1))=1;
8   end
9   xe = pdist(data,'euclidean');
10  x = squareform(xe);
11  thresholdx=[];
12  rentang = max(mst(:,3))- min(mst(:,3));
13  step_size = rentang * step_size +
    min(mst(:,3));
14  threshold = rentang * threshold +
    min(mst(:,3));
15  while(signal == 0)
16  index=zeros(N,1);
17      counter1=1;
18      T=A; Cluster_no=0.0;
19      % storing end points with edge weight >
    threshold and removing that edge from MST
20      Storage=zeros(2*(N-1),d+1);
21      counter=0;
```

Kode Sumber 4.10. Kode Sumber klastering berdasarkan MST

```
1   for i=1:N-1
2           for j=i+1:N
3               if(T(i,j) == 1)
4                   if(sqrt(sum((sorted(i,:)-
    sorted(j,:)).^2)) < threshold)
5                       counter = counter+1;
6                       E(counter,:) = [i j];
7                   else
8                       T(i,j)=0;T(j,i)=0;
9
    Storage(counter1,:)=[sorted(i,:) i];
10
    Storage(counter1+1,:)=[sorted(j,:) j];
11                      counter1=counter1+2;
12                  end
13              end
14          end
15      end
16  Cluster_no=(counter1+1)  /  2;  %  Cluster_no
    stores the number of cluster center
17      counter1=1;
18      if(Storage(counter1,d+1) > 0) % more then
    one cluster
19          % Finding Index of each sorted point
20          counterx = 1;
21          index=zeros(N,1);
22          for i=1:size(E,1)
23              if E(i,1)~=0
24                  if    index(E(i,1))==0    &&
    index(E(i,2))==0
25                      index(E(i,1))=counterx;
26                      index(E(i,2))=counterx;
27                      counterx=counterx+1;
28                  elseif index(E(i,1))==0
29
    index(E(i,1))=index(E(i,2));
30                  elseif index(E(i,2))==0
31
    index(E(i,2))=index(E(i,1));
32                  else
```

Kode Sumber 4.11. Kode Sumber klastering berdasarkan MST (2)

```
1                       if    index(E(i,1))    > index(E(i,2))
2                           index(E(i,1))      = index(E(i,2));
3                       elseif  index(E(i,2))  > index(E(i,1))
4                           index(E(i,2))      = index(E(i,1));
5                       end
6                   end
7               end
8           end
9           for i=1:size(E,1)
10              index(E(i,1))= min(index(E(i,1)),index(E(i,2)));
11              index(E(i,2))= min(index(E(i,1)),index(E(i,2)));
12          end
13          for i=1:N
14              if(index(i)==0)
15                  index(i)=counterx;
16                  counterx = counterx+1;
17              end
18          end
19          while(          Cluster_no          ~= size(unique(index)))
20              for i=1:size(E,1)
21                  index(E(i,1))= min(index(E(i,1)),index(E(i,2)));
22                  index(E(i,2))= min(index(E(i,1)),index(E(i,2)));
23              end
24          end
25          cluster =unique(index);
26
xc=zeros(Cluster_no,d);ff=0;summation=0;    % xc stores the cluster center
27          for i = 1:Cluster_no
28              Ind = find(index == cluster(i));
29              if(length(Ind) == 1)
```

Kode Sumber 4.12. Kode Sumber klastering berdasarkan MST (3)

```
                summation=summation + (max(xe))^2;
                %summation=summation + (max(PV(:,3)))^2;
            end
        end
        Inter=(min(pdist(xc)))^2;
        D=zeros(Cluster_no,N);
for i=1:N
            for k=1:Cluster_no
                D(k,i) = (sum(xc(k,:)-sorted(i,:)).^2);
            end
        end
        Intra=(sum(min(D)) + summation) / N;
        ratio(1,it)=Intra / Inter;

        C(1,it) = Cluster_no;
        thresholdx(1,it)=threshold;
        it=it+1;
        threshold=threshold + step_size;
    else % only one cluster is there
        Intra=0.0;xc=zeros(1,d);
        xc(1,:)=mean(sorted(:,:));
        for i=1:N
            Intra=Intra+(sum(xc(1,:)-sorted(i,:)).^2);
        end
        Intra=Intra/N;
        Inter=min(xe);
%         Inter=min(PV(:,3));
        if(Inter == 0)
            Inter=0.1;
        end
        C(1,it) = Cluster_no;
        ratio(1,it)=Intra / Inter;
        thresholdx(1,it)=threshold;
        signal =1;
    end
end
```

Kode Sumber 4.13. Kode Sumber klastering berdasarkan MST (4)

4.2.8 Implementasi Evaluasi Index Dunn

Index Dunn merupakan salah satu evaluasi internal hasil klastering. Evaluasi ini ditujukan untuk mencari agar klaster-klaster yang terbentuk memiliki kemiripan yang tinggi dalam satu klaster dan kemiripan yang rendah dalam klaster yang berbeda. Indeks ini merupakan rasio dari jarak terpendek titik beda klaster dengan jarak terpanjang titik yang terletak pada satu klaster. Semakin besar indeks Dunn yang dihasilkan, maka hasil klaster dinyatakan semakin baik. Implementasi Indeks Dunn ditunjukkan pada Kode Sumber 4.14.

```
1   function [DI num
    dem]=dunnsM(clusters_number,distM,ind)
2   i=clusters_number;
3   denominator=[];
4   cluster=unique(ind);
5   for i2=1:i
6       indi=find(ind==cluster(i2));
7       indj=find(ind~=cluster(i2));
8       x=indi;
9       y=indj;
10      temp=distM(x,y);
11      denominator=[denominator;temp(:)];
12  end
13  num=min(min(denominator));
14  neg_obs=zeros(size(distM,1),size(distM,2));
15
16  for ix=1:i
17      indxs=find(ind==cluster(ix));
18      neg_obs(indxs,indxs)=1;
19  end
20
21  dem=neg_obs.*distM;
22  dem=max(max(dem));
23
24  DI=num/dem;
25  end
```

Kode Sumber 4.14. Kode Sumber Indeks Dunn

[Halaman ini sengaja dikosongkan]

BAB V
UJI COBA DAN EVALUASI

Pada bab ini akan dijelaskan uji coba yang dilakukan pada aplikasi yang telah dikerjakan serta analisa dari uji coba yang telah dilakukan. Pembahasan pengujian meliputi lingkungan uji coba, skenario uji coba yang meliputi uji kebenaran dan uji kinerja serta analisa setiap pengujian.

5.1 Lingkungan Uji Coba

Lingkungan uji coba menjelaskan lingkungan yang digunakan untuk menguji implementasi *Fast Minimum Spanning Tree (FMST)*, dan klastering berdasarkan *Minimum Spanning Tree (MST)* pada Tugas Akhir ini. Lingkungan uji coba meliputi perangkat keras dan perangkat lunak yang dijelaskan sebagai berikut:

1. Perangkat keras
   a. Prosesor: Intel® Core ™ i5-2410M CPU @ 2.30GHz (4 CPUs), ~2.3GHz
   b. *Memory*(RAM): 4,00 GB
   c. Tipe sistem: 64-bit sistem operasi
2. Perangkat lunak
   a. Sistem operasi: *Windows* 8.1. Pro
   b. Perangkat pengembang: *MATLAB 2008*.

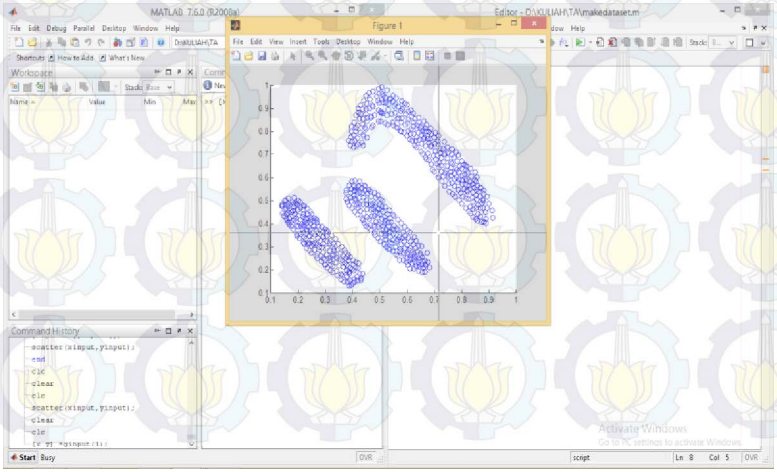
5.2 Data Uji Coba

Data yang digunakan merupakan data sintesis dan data real. Data sintesis adalah data yang didesain menyerupai situasi tertentu, yang mungkin tidak ditemui dalam data real. Dalam Tugas akhir ini, akan dibentuk data sintesis yang menyerupai bentuk-bentuk tertentu yang bersifat *non-Convex*.

Data-data ini dibuat secara manual oleh penulis. Data dibuat menggunakan *MATLAB* 2008 memanfaatkan fungsi yang sudah tersedia, yaitu *ginput()*. Perintah *ginput()* merupakan cara untuk memilih titik-titik dari grafik aktif dengan bantuan *mouse*. *[x,y]=ginput(n)* merupakan fungsi yang mengambil n titik dari sumbu aktif dan mengisikan koordinatnya dalam *array* kolom x dan y [20]. Contoh Kode Sumber untuk membuat data sintesis dapat dilihat pada Kode Sumber 5.1. Proses pembuatan *dataset* dapat dilihat pada Gambar 5.1.

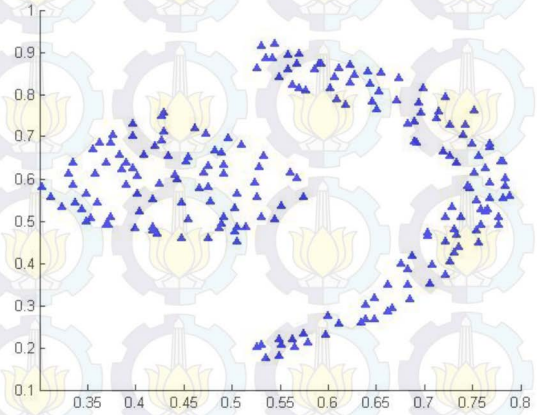
```
1.  ylim([0 1]);
2.  xlim([0 1]);
3.  xinput=[];
4.  yinput=[];
5.  scatter(xinput,yinput);
6.
7.  for i=1:50
8.      [x y] =ginput(1);
9.      xinput = [xinput x];
10.     yinput = [yinput y];
11.     scatter(xinput,yinput);
12. end
```

Kode Sumber 5.1. Kode Sumber membuat Data Sintesis

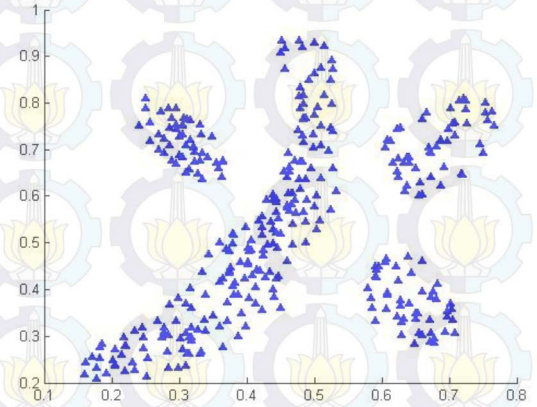


Gambar 5.1. Proses pembuatan data sintesis

Visualisasi data sintesis ini dapat dilihat pada Gambar 5.2, dan Gambar 5.3. Visualisasi data lebih lengkap dapat dilihat pada. LAMPIRAN. Data real yang dipakai merupakan *dataset iris* dan *wine* [21].



Gambar 5.2. Visualisasi Dataset T1



Gambar 5.3. Visualisasi Dataset T3

5.3 Skenario dan Evaluasi Pengujian

Uji coba ini dilakukan untuk menguji apakah fungsionalitas program telah diimplementasikan dengan benar dan berjalan sebagaimana mestinya. Uji coba akan didasarkan pada beberapa skenario untuk menguji kesesuaian dan kinerja aplikasi.

Pengujian terdiri dari 3 pengujian yaitu:

1. Perhitungan waktu dalam pembuatan *FMST dan MST*.
2. Perhitungan *weight error* yang didapat dalam pembuatan *FMST*.
3. Uji hasil klastering yang diperoleh dari klastering berdasarkan *FMST*, *MST*, dan K-Means.

5.3.1 Skenario Uji Coba dengan Data Sintesis

Uji coba dilakukan sebanyak 15 kali yang terdiri dari 15 data sintesis yang berbeda. Detil tiap percobaan dapat dilihat pada Tabel 5.1. *FMST* dilakukan 5 kali pada tiap data. Pada Gambar 5.4 ditunjukkan bahwa garis hijau adalah waktu rata-rata yang diperlukan metode *FMST* pada 5 kali percobaan. Sedangkan garis biru adalah waktu yang diperlukan untuk metode *FMST* dengan hasil yang terbaik pada 5 kali percobaan.

Uji coba juga dilakukan dengan menggunakan algoritma konvensional untuk menyelesaikan *MST*. Hasil waktu yang diperlukan oleh algoritma ini dapat dilihat pada Gambar 5.5.

Setelah mendapatkan hasil *weight* yang merupakan jumlah dari *edge-edge* yang diperoleh dari *FMST* dan algoritma konvensional *MST*, maka dilakukan perhitungan *weight error* yang menggunakan rumus:

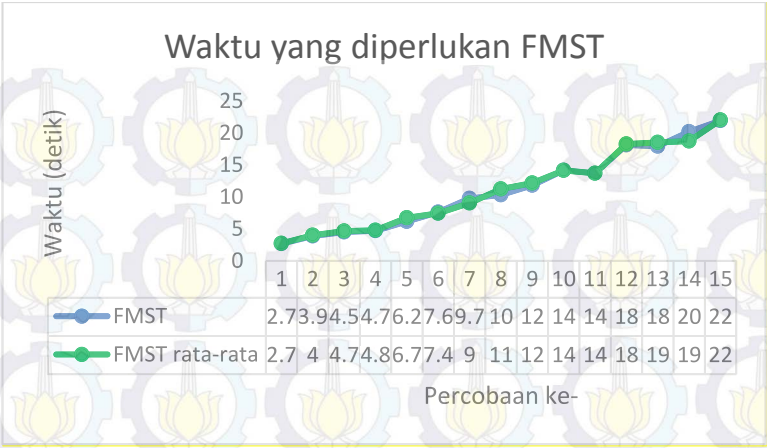
$$Weight\ error = \frac{weight(fmst) - weight(mst)}{weight(mst)} \quad (5.1)$$

Weight error = nilai *weight error*
Weight(fmst) = jumlah dari bobot *edge* pada *MST* yang dihasilkan *FMST*
Weight(mst) = jumlah dari bobot *edge* pada *MST* yang dihasilkan *MST* sebenarnya

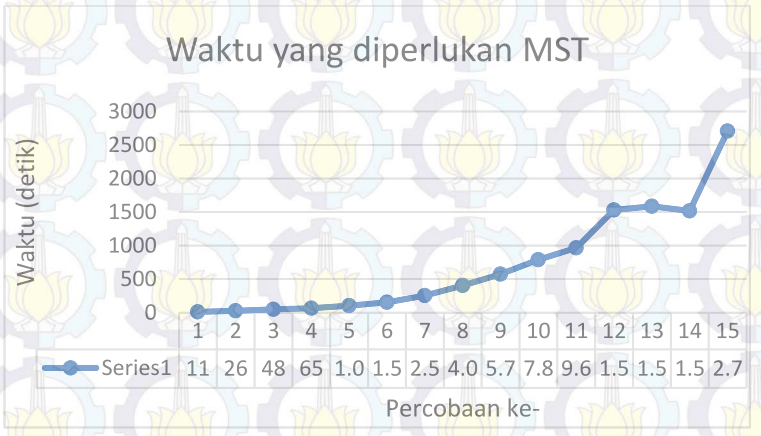
MST yang dihasilkan metode *FMST* dinilai baik jika memiliki *weight* yang sama dengan *MST* aslinya. Maka dari itu, semakin kecil *weight error*, maka *MST* yang dihasilkan metode *FMST* dinilai semakin baik.

Tabel 5.1. Tabel Detail Data Sintesis tiap Percobaan

Percobaan ke-	*Dataset* yang dipakai	Ukuran data (jumlah x fitur)	Jumlah klaster
1	T1	200 x 2	2
2	T2	250 x 2	3
3	Curve	300 x 2	2
4	Spiral1	312 x 2	3
5	T3	350 x 2	4
6	T4	400 x 2	5
7	T5	450 x 2	5
8	T6	500 x 2	5
9	T7	550 x 2	5
10	T8	600 x 2	2
11	Spiral2	612 x 2	3
12	T9	700 x 2	2
13	T10	700 x 2	3
14	T11	700 x 2	3
15	T12	800 x 2	4



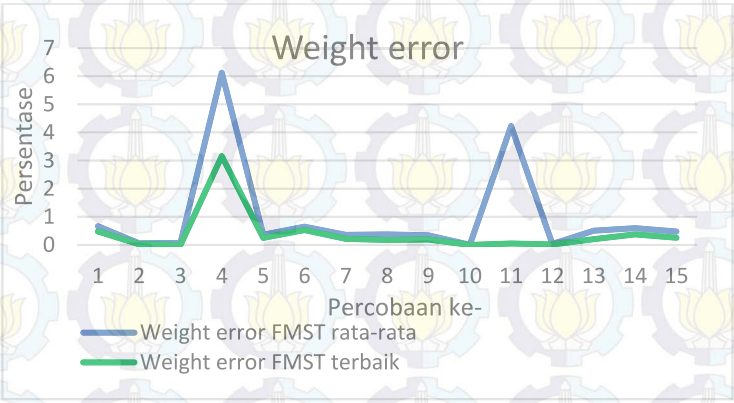
Gambar 5.4. Grafik Waktu yang diperlukan FMST



Gambar 5.5. Grafik Waktu yang diperlukan algoritma Prim

Hasil perhitungan *weight error* dapat dilihat pada Gambar 5.6. Pada Gambar 5.6 ditunjukkan bahwa garis biru adalah *weight error* rata-rata yang dihasilkan metode *FMST* pada 5 kali

percobaan. Sedangkan garis hijau adalah *weight error* terbaik yang dihasilkan untuk metode *FMST* dalam 5 kali percobaan.



Gambar 5.6. Grafik weight error

Setelah itu, dilakukan klastering berdasarkan *MST* yang didapat dari metode *FMST* dan algoritma konvensional *MST*, dan juga menggunakan algoritma klastering K-Means. Parameter jumlah klaster yang dipakai algoritma K-Means adalah jumlah klaster yang dihasilkan oleh klastering berdasarkan algoritma konvensional *MST*. Hasil klastering yang diperoleh dievaluasi menggunakan Indeks Dunn. Skenario Uji Coba dilakukan dengan *threshold* bernilai 0.1 dan *step_size* bernilai 0.1, 0.05, dan 0.025. Setelah dilakukan uji coba, ternyata hasil percobaan menggunakan *threshold*= 0.1, *step_size*= 0.1 dan 0.05 adalah sama. Hasil uji coba menggunakan *threshold*= 0.1, *step_size*= 0.1 dan 0.05 dapat dilihat pada Tabel 5.2. Hasil uji coba menggunakan *threshold*= 0.1, *step_size*= 0.025 dapat dilihat pada Tabel 5.3. Kolom yang diberi warna hijau adalah kolom yang memuat nilai indeks Dunn lebih besar dibanding kolom lainnya.

Tabel 5.2. Tabel Hasil Uji Coba 1 (threshold = 0.1, step_size =0.1 dan 0.05)

Percobaan ke-	Indeks Dunn *FMST*	Indeks Dunn *MST*	Indeks Dunn Kmeans	Jumlah Klaster yang dihasilkan *FMST*	Jumlah Klaster yang dihasilkan *MST*
1	0.184	0.184	0.033	2	2
2	0.183	0.183	0.045	2	2
3	0.232	0.232	0.014	2	2
4	0.036	0.141	0.01	4	3
5	0.105	0.105	0.058	4	4
6	0.474	0.474	0.474	5	5
7	0.085	0.085	0.022	5	5
8	0.085	0.085	0.034	5	5
9	0.085	0.085	0.029	5	5
10	0.257	0.257	0.257	2	2
11	0.134	0.134	0.006	3	3
12	0.123	0.123	0.012	2	2
13	0.175	0.175	0.022	3	3
14	0.203	0.203	0.203	2	2
15	0.156	0.156	0.022	4	4

Tabel 5.3. Tabel Hasil Uji Coba 2 (threshold = 0.1, step_size =0.025)

Percobaan ke-	Indeks Dunn *FMST*	Indeks Dunn *MST*	Indeks Dunn Kmeans	Jumlah Klaster yang dihasilkan *FMST*	Jumlah Klaster yang dihasilkan *MST*
1	0.184	0.184	0.033	2	2
2	0.183	0.183	0.045	2	2
3	0.053	0.053	0.056	6	6
4	0.025	0.043	0.016	3	4
5	0.105	0.105	0.058	4	4
6	0.474	0.474	0.474	5	5
7	0.085	0.085	0.022	5	5
8	0.085	0.085	0.034	5	5
9	0.085	0.085	0.029	5	5
10	0.257	0.257	0.257	2	2
11	0.134	0.134	0.006	3	3
12	0.123	0.123	0.012	2	2
13	0.175	0.175	0.022	3	3
14	0.203	0.203	0.203	2	2
15	0.156	0.156	0.022	4	4

5.3.2 Skenario Uji Coba dengan Beberapa *Dataset*

Uji coba dilakukan terhadap dua *dataset* yang umum digunakan yaitu, data *iris* dan data *wine*. Uji coba dilakukan sebanyak 20 kali pada tiap data. Waktu dan *weight error* yang didapat pada uji coba tercatat pada Tabel 5.4. Setelah itu, dilakukan klastering berdasarkan *MST* yang didapat dari metode *FMST* dan algoritma konvensional *MST*, dan juga menggunakan algoritma

klastering K-Means. Parameter jumlah klaster yang dipakai algoritma K-Means adalah jumlah klaster yang dihasilkan oleh klastering berdasarkan algoritma *MST* konvensional. Hasil klastering yang diperoleh dievaluasi menggunakan Indeks Dunn. Skenario Uji Coba dilakukan dengan *threshold* bernilai 0.1 dan *step_size* bernilai 0.1, 0.05, dan 0.025. Setelah dilakukan uji coba, ternyata hasil percobaan menggunakan *threshold*= 0.1, *step_size*= 0.1, 0.05, dan 0.025 adalah sama. Hasil uji coba tersebut dapat dilihat pada Tabel 5.5. Pada Tabel 5.5, kolom yang diberi warna hijau adalah kolom yang memuat nilai indeks Dunn lebih besar dibanding kolom lainnya. Pada Tabel 5.5 baik data *iris* maupun *wine*, klastering yang dilakukan menghasilkan 2 klaster, sesuai dengan nilai Indeks Dunn terbesar yang didapatkan. Sedangkan, menurut *ground truth* nya, kedua *dataset* tersebut terdiri dari 3 klaster. Maka dari itu dilakukan skenario coba dilakukan evaluasi klastering menggunakan klastering *FMST*, *MST* dan Kmeans dengan "memaksakan" data dipartisi menjadi 3 klaster. Nilai evaluasi tersebut dapat dilihat pada Tabel 5.6.

Selain Indeks Dunn, hasil klastering juga dinilai melalui *ground truth.* Label klaster yang dihasilkan oleh klastering berdasarkan *FMST*, klastering berdasarkan *MST*, K-Means dicocokkan dengan label klaster *ground truth.* Hasil perbandingan tersebut dapat di lihat pada Tabel 5.7, Tabel 5.8, Tabel 5.9, dan Tabel 5.10. Pada tabel – tabel tersebut, data dikelompokkan menurut hasil klastering prediksi dan hasil klastering sebenarnya. Sebagai contoh, pada Tabel 5.8, dengan menggunakan klastering berdasarkan *FMST*, *dataset Iris* dibagi menjadi 3 klaster. Klaster prediksi 1 memuat 50 data yang semuanya berasal dari klaster 1 yang sebenarnya. Klaster prediksi 2 memuat 98 data, terdiri dari 50 data berasal dari klaster 2 sesungguhnya dan 48 data yang berasal

dari klaster 3 sesungguhnya. Klaster prediksi 3, terdiri dari 2 data yang seluruhnya berasal dari klaster 3 sesungguhnya.

Tabel 5.4. Tabel Perbandingan Waktu dan weight error

	Waktu yang diperlukan *FMST* (hasil terbaik)	**Waktu rata-rata yang diperlukan *FMST***	**Waktu yang diperlukan *MST***	***Weight error* minimum**	***Weight error* rata-rata**
Iris	2.088	1.787	4.972	0	0.426
Wine	2.073	2.166	7.944	0	0.074

Tabel 5.5. Tabel Hasil Uji Coba 1 terhadap Data Real

	Indeks Dunn *FMST*	**Indeks Dunn *MST***	**Indeks Dunn K-Means**	**Jumlah Klaster yang dihasil-kan *FMST***	**Jumlah Klaster yang dihasil-kan *MST***
Iris	0.339	0.339	0.077	2	2
Wine	0.105	0.105	0.023	2	2

Tabel 5.6. Tabel Hasil Uji Coba 2 terhadap Data Real

	Indeks Dunn *FMST*	Indeks Dunn *MST*	Indeks Dunn K-Means
Iris	0.169	0.169	0.099
Wine	0.068	0.068	0.016

Tabel 5.7. Tabel Hasil klastering pada Uji Coba 1 (Dataset Iris)

Klaster sebenarnya	*Ground Truth*	Klaster prediksi					
		FMST		*MST*		K-Means	
		1	2	1	2	1	2
1	50	50	0	50	0	50	0
2	50	0	50	0	50	3	47
3	50	0	50	0	50	0	50

Tabel 5.8. Tabel Hasil klastering pada Uji Coba 2 (Dataset Iris)

Klaster Sebenar-nya	*Ground Truth*	Klaster prediksi								
		FMST			*MST*			K-Means		
		1	2	3	1	2	3	1	2	3
1	50	50	0	0	50	0	0	50	0	0
2	50	0	50	0	0	50	0	0	48	2
3	50	0	48	2	0	48	2	0	14	36

Tabel 5.9. Tabel Hasil klastering pada Uji Coba 1 (Dataset Wine)

Klaster Sebenar nya	*Ground Truth*	Klaster prediksi					
		FMST		*MST*		Kmeans	
		1	2	1	2	1	2
1	59	1	58	1	58	50	9
2	71	0	71	0	71	4	67
3	48	0	48	0	48	1	47

Tabel 5.10. Tabel Hasil klastering pada Uji Coba 2 (Dataset Wine)

Klaster sebenarnya	*Ground Truth*	Klaster prediksi								
		FMST			*MST*			K-Means		
		1	2	3	1	2	3	1	2	3
1	59	5	53	1	5	53	1	46	0	13
2	71	0	71	0	0	71	0	1	50	20
3	48	0	48	0	0	48	0	0	19	29

5.4 Analisis Hasil Uji Coba

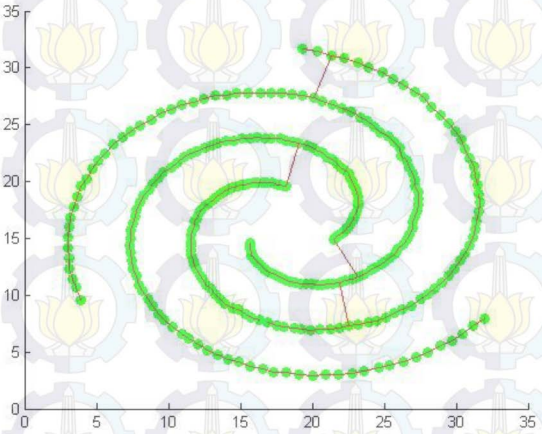
Analisa dilakukan tersendiri pada data sintesis dan data real. Analisa dilakukan berdasarkan hasil dari tiap skenario uji coba dengan evaluasi tiap pengujian.

5.4.1 Analisa Hasil Uji Coba Data Sintesis

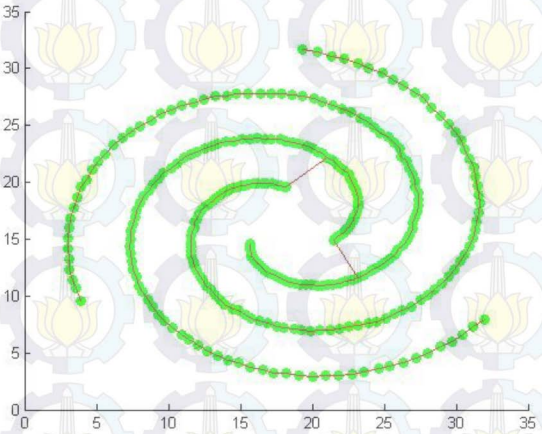
Berdasarkan Gambar 5.1 dan Gambar 5.2, diketahui bahwa metode *FMST* dapat menyelesaikan permasalahan *MST* dengan waktu yang lebih sedikit. Untuk masalah akurasi, dapat dilihat pada Gambar 5.3, bahwa 13 dari 15 percobaan memiliki *weight error* dibawah 1 persen. Hasil percobaan ke 4 dan ke 11 pada data sintesis yang memiliki *weight error* rata-rata diatas 1 persen dikarenakan kelompok-kelompok pada data tersebut tidak terpisah jauh. Visualisasi *FMST* pada percobaan ke 4 dapat dilihat pada Gambar 5.7, serta *MST* sebenarnya pada percobaan ke 4 dapat dilihat pada Gambar 5.8. Terlihat perbedaan mencolok pada 2 gambar tersebut.

Percobaan ke 3, 7, dan 11 memiliki *weight error* rata-rata hampir mendekati 0 persen. Hal ini dikarenakan kelompok-kelompok pada data tersebut terpisah secara baik. Visualisasi *FMST* pada percobaan ke 3 dapat dilihat pada Gambar 5.9, serta

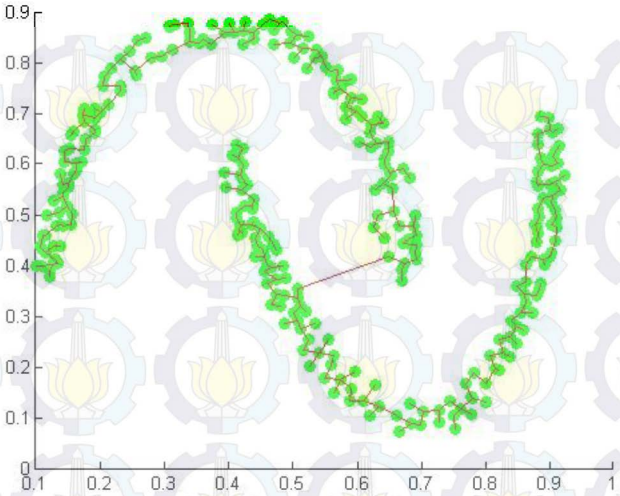
MST sebenarnya pada percobaan ke 3 dapat dilihat pada Gambar 5.10. Tidak ada perbedaan diantara 2 gambar tersebut.



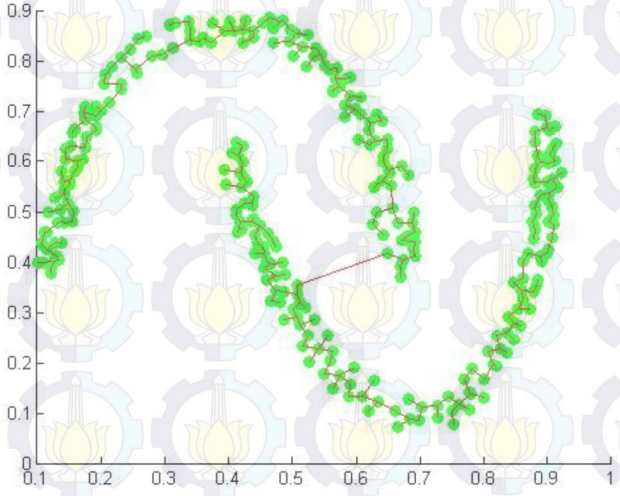
Gambar 5.7. Hasil FMST terbaik pada percobaan ke 4



Gambar 5.8. Hasil MST pada percobaan ke 4

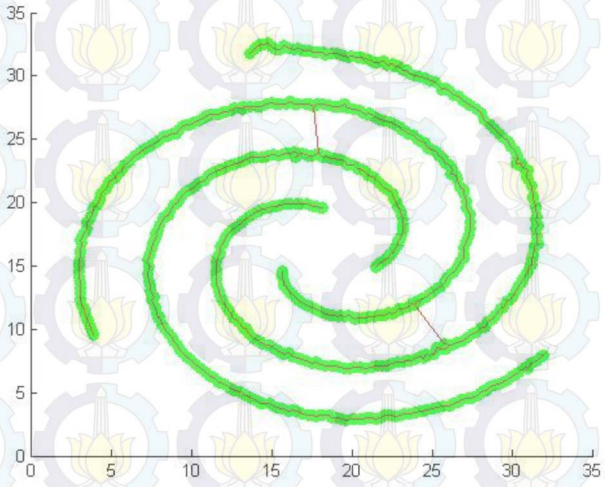


Gambar 5.9. Hasil FMST terbaik pada percobaan ke 3

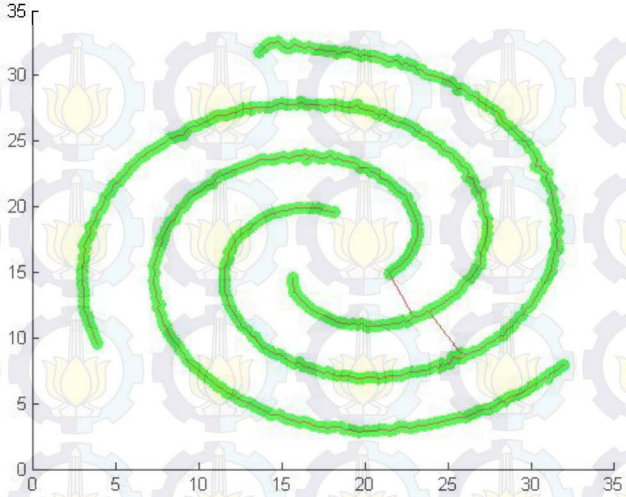


Gambar 5.10. Hasil MST pada percobaan ke 3

Kerapatan antar data juga menjadi faktor penting agar metode *FMST* dapat menyelesaikan permasalahan *MST* dengan baik. Hal ini dapat dilihat pada percobaan ke 4 dan ke 11 pada data sintesis, data pada percobaan tersebut memiliki bentuk yang sama, hanya jumlah data yang berbeda. Percobaan ke 4 yang memiliki 312 data menghasilkan *weight error* lebih tinggi dibanding dengan percobaan ke 11 yang memiliki 612 data. Semakin banyak data, maka semakin besar kelompok yang dihasilkan pada tahap *Divide and Conquer*. Semakin besar kelompok pada tahap *Divide and Conquer*, maka semakin besar pula kemungkinan *subset* yang dihubungkan dengan *subset* yang benar pada tahap *Combine Subset Algorithm*. Visualisasi *FMST* pada percobaan ke 11 dapat dilihat pada Gambar 5.11, serta *MST* sebenarnya pada percobaan ke 11 dapat dilihat pada Gambar 5.12.



Gambar 5.11. Hasil FMST terbaik pada percobaan ke 11

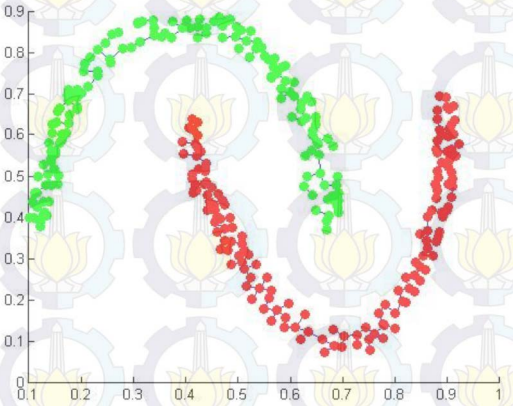


Gambar 5.12. Hasil MST pada percobaan ke 11

Dilihat dari Tabel 5.1, Tabel 5.2, dan Tabel 5.3, skenario uji coba dengan menggunakan *threshold= 0.1* dan *step_size= 0.05* atau *0.1* menghasilkan hasil klastering lebih baik dibandingkan skenario uji coba menggunakan *threshold= 0.1* dan *step_size= 0.025*. Hal ini ditunjukkan dari 15 percobaan menggunakan *step_size= 0.05* atau *0.1,* 13 percobaan diantaranya menunjukkan jumlah klaster yang dihasilkan klastering berdasarkan *MST* sama dengan jumlah klaster sebenarnya. Percobaan pada uji coba 1 yang gagal mengidentifikasi jumlah klaster adalah percobaan 2, dan 14. Sedangkan, jika menggunakan *step_size= 0.025,* hanya 11 dari 15 percobaan yang menunjukkan jumlah klaster yang dihasilkan klastering berdasarkan *MST* sama dengan jumlah klaster sebenarnya. Percobaan pada uji coba 2 yang gagal mengidentifikasi jumlah klaster adalah percobaan 2, 3, 4, dan 14.

Ditinjau dari Nilai Indeks Dunn, Tabel 5.2, dan Tabel 5.3 didominasi dengan lebih bagusnya hasil klastering berdasarkan *MST* dibanding Kmeans. Satu-satunya percobaan yang

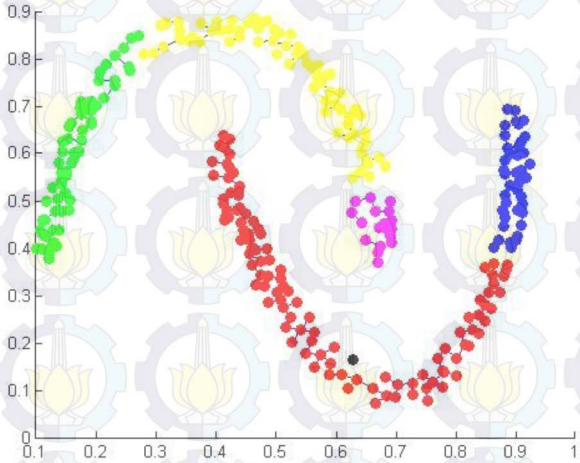
membedakan adalah pada percobaan ke 3, dimana pada uji coba ke 2 untuk data sintesis menunjukkan hasil klastering K-Means menghasilkan indeks Dunn lebih besar. Visualisasi tersebut dapat dilihat pada Gambar 5.15.Visualisasi *FMST* pada uji coba 1 untuk percobaan ke 3 pada data sintesis dapat dilihat pada Gambar 5.13, serta *MST* sebenarnya pada uji coba 2 untuk percobaan ke 3 dapat dilihat pada Gambar 5.14.



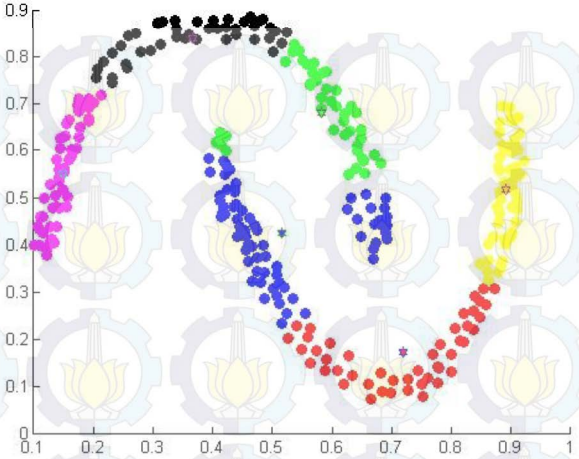
Gambar 5.13. Hasil klastering menggunakan FMST pada uji coba 1 untuk percobaan ke 3

Dua percobaan yang sama-sama gagal diidentifikasi Tabel Hasil Uji Coba 1 (*threshold* = 0.1, *step_size* =0.1 dan 0.05), dan Tabel Hasil Uji Coba 2 (*threshold* = 0.1, *step_size* =0.025) dengan benar dalam hal jumlah klaster nya adalah percobaan ke 2 dan ke 14. Percobaan tersebut, jumlah klasternya gagal diidentifkasi dengan benar menggunakan klastering berdasarkan *FMST*, maupun berdasarkan *MST*. Jumlah klaster yang sebenarnya pada *dataset* adalah 3, namun hanya diidentifikasi terdiri dari 2 klaster. Hal ini mungkin diakibatkan klaster – klaster yang ada tidak terletak begitu jauh. Sebagai argumen, percobaan ke 13 pada data

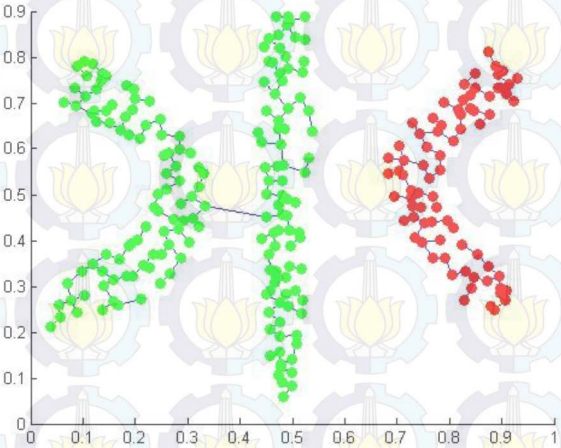
sintesis memiliki bentuk *dataset* yang sama dengan percobaan ke 14 pada data sintesis. Jumlah data pada *dataset* pun sama-sama 700. Namun *dataset* percobaan ke 13, klaster-klasternya terletak lebih berjauhan dibanding percobaan ke 14. Sehingga percobaan ke 13 mengembalikan jumlah klaster yang tepat dengan jumlah klaster sebenarnya, yaitu 3. Visualisasi hasil klastering berdasarkan *FMST*/ *MST* percobaan ke 2 dapat dilihat pada Gambar 5.16. Visualisasi kasil klastering berdasarkan *FMST*/ *MST* percobaan ke 14 dapat dilihat pada Gambar 5.17. Visualisasi hasil klastering berdasarkan *FMST* / *MST* percobaan ke 13 dapat dilihat pada Gambar 5.18.



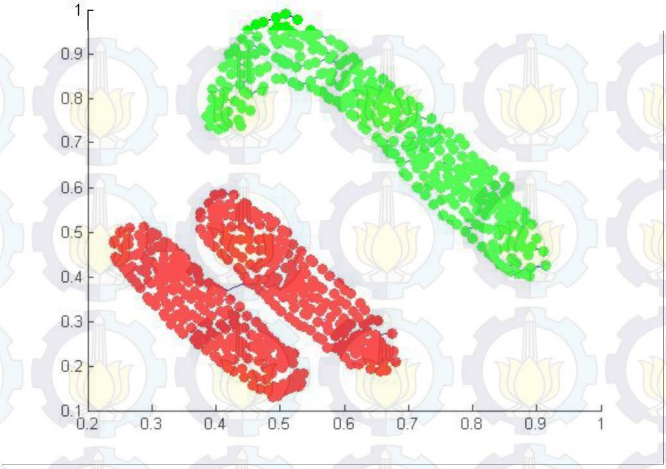
Gambar 5.14. Hasil klastering menggunakan FMST pada uji coba 2 untuk percobaan ke 3



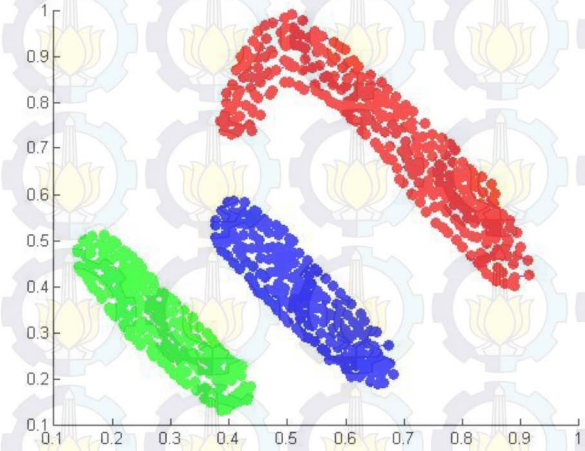
Gambar 5.15. Hasil klastering menggunakan Kmeans pada uji coba 2 untuk percobaan ke 3



Gambar 5.16. Hasil klastering menggunakan FMST/MST pada percobaan ke 2

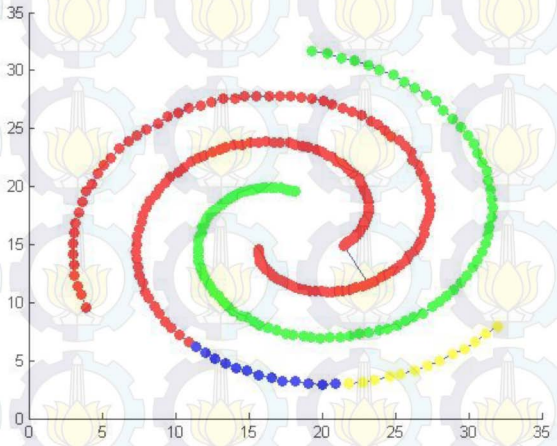


Gambar 5.17. Hasil klastering menggunakan FMST/MST pada percobaan ke 14



Gambar 5.18. Hasil klastering menggunakan FMST/MST pada percobaan ke 13

Klastering berdasarkan *FMST* tidak kalah baik dibanding klastering berdasarkan *MST*. Dapat dilihat pada Tabel 5.2, dan Tabel 5.3, klastering berdasarkan *FMST* kalah dibanding klastering berdasarkan *MST* di percobaan ke 4. Visualisasi hasil klastering berdasarkan *FMST*/ *MST* percobaan ke 4 dapat dilihat pada Gambar 5.19. Hal ini ditengarai akibat *weight error* percobaan ke 4 bernilai relatif tinggi.



Gambar 5.19. Hasil klastering menggunakan FMST pada percobaan ke 4 uji coba 1

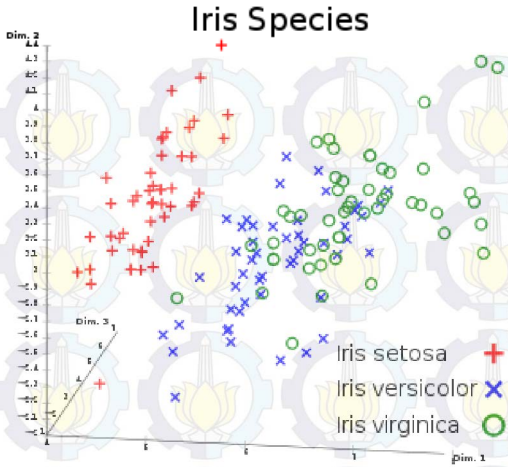
5.4.2 Analisa Hasil Uji Coba Data Real

Menurut Tabel 5.5, hasil indeks Dunn klastering menggunakan *FMST*, maupun *MST* pada *dataset iris*, dan *wine* bernilai lebih besar dibanding hasil klastering K-Means. Hal ini dapat dilihat pula pada Tabel 5.6, dimana uji coba dilakukan menggunakan jumlah klaster yang sama jumlah klaster sesungguhnya, yaitu 3 klaster. Pada tabel tersebut, hasil indeks Dunn klastering menggunakan *FMST*, maupun *MST* pada *dataset iris* dan *wine* tetap bernilai besar dari hasil K-Means. Kedua uji

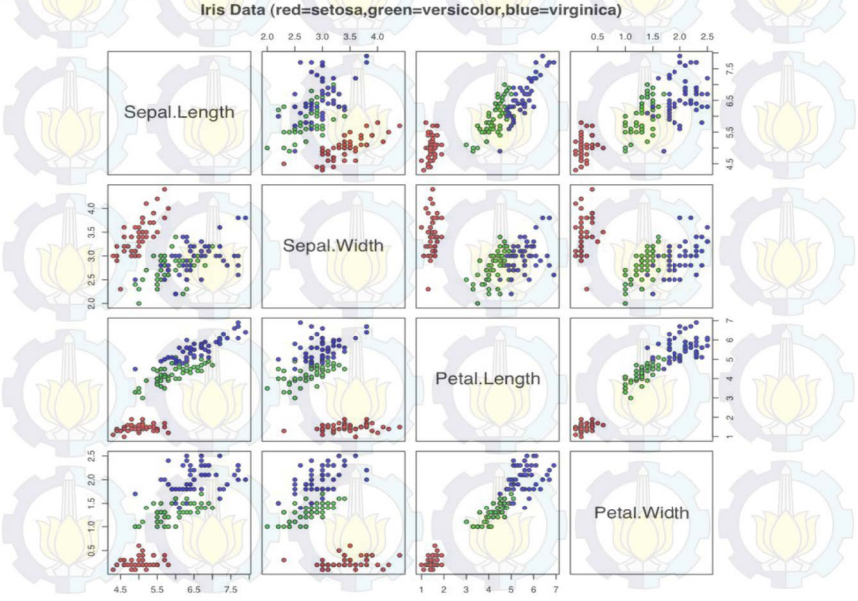
coba tersebut menandakan bahwa menggunakan klastering berdasarkan *MST* lebih baik dibanding menggunakan K-Means.

Akan, tetapi, K-Means terbilang lebih baik dalam hal mengelompokkan data ke klaster yang sesuai dengan klaster sebenarnya. Hal ini dapat dilihat pada Tabel 5.8, dan Tabel 5.10. Pada Tabel 5.8, klastering berdasarkan *FMST*, maupun *MST* hanya dapat mengelompokkan 102 data ke klaster yang tepat. Sedangkan K-Means mampu mengelompokkan 134 data ke klaster yang tepat. Pada Tabel 5.10, klastering berdasarkan *FMST*, maupun *MST* hanya dapat mengelompokkan 76 data ke klaster yang tepat. Sedangkan K-Means mampu mengelompokkan 125 data ke klaster yang tepat. Hal ini mungkin disebabkan dataset Iris dan Wine memiliki *overlapping* yang tinggi. *FMST/ MST* berhasil mengelompokkan *dataset* Iris dengan baik ketika *dataset* dibagi menjadi 2 klaster. Hal ini dapat dilihat pada Tabel 5.7, bahwa hasil klastering menggunakan *FMST/ MST* lebih baik daripada K-Means. Sedangkan dengan *dataset* Wine pada Tabel 5.8, ketika diklastering menggunakan *FMST/ MST* tidak mengelompokkan data lebih baik dibanding menggunakan K-Means.

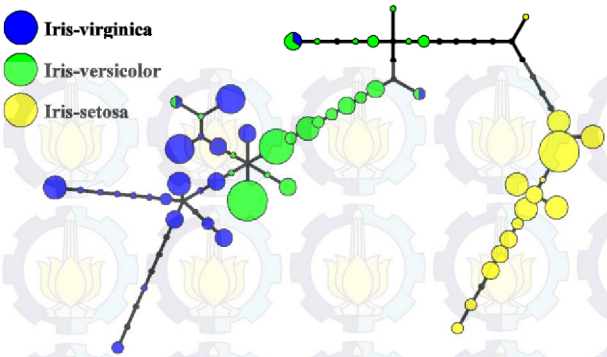
Dapat dilihat pada Gambar 5.20, Gambar 5.21, dan Gambar 5.22 [22], *dataset* Iris di-*plot* ke beberapa macam representasi. Kelas *iris-setosa* memiliki tingkat kemiripan yang sangat jauh dengan dua kelas lainnya. Sedangkan kelas *iris-versicolor*, dan kelas *iris-virginica* saling tumpah tindih (*overlap*). Sehingga ketika dataset Iris dilakukan klastering menggunakan *FMST/ MST*, meskipun terbentuk klaster yang memiliki *separation* baik (ditandai dengan nilai indeks Dunn yang relatif tinggi dibanding K-Means), dataset Iris tidak terkelompokkan secara tepat.



Gambar 5.20. Ilustrasi dataset Iris



Gambar 5.21. Ilustrasi dataset Iris dibandingkan tiap atribut



Gambar 5.22. Ilustrasi "metromap" dataset Iris

[Halaman ini sengaja dikosongkan]

BAB VI
KESIMPULAN DAN SARAN

Bab ini membahas mengenai kesimpulan yang dapat diambil dari hasil uji coba yang telah dilakukan sebagai jawaban dari rumusan masalah yang dikemukakan. Selain kesimpulan, juga terdapat saran yang ditujukan untuk pengembangan penelitian lebih lanjut.

6.1 Kesimpulan

Dari hasil uji coba yang telah dilakukan terhadap pembuatan model, dapat diambil kesimpulan sebagai berikut:

1. Metode *Fast Minimum Spanning Tree* (*FMST*) dapat digunakan untuk menyelesaikan *MST* lebih cepat dibanding algoritma *MST* konvensional dengan *weight error* rata-rata 0.988%.
2. Klastering berdasarkan *FMST* mendapatkan hasil terbaik ketika menggunakan *threshold*= 0.1 dan *step_size*= 0.1.
3. Klastering menggunakan *FMST* mampu mengelompokkan dataset yang berbentuk n*on-convex.* Akan lebih optimal jika data memiliki jarak antar klaster dan kerapatan data cukup tinggi. Akan tetapi, *FMST* kurang optimal digunakan mengelompokkan data yang memiliki *overlap* tinggi.

6.2 Saran

Saran yang diberikan untuk pengembangan aplikasi ini adalah:

1. Untuk meningkatkan waktu pemrosesan, dapat digunakan bahasa pemrograman yang lain.
2. Perlu penelitian lebih lanjut mengenai tahap *Combine Subset Algorithm* untuk meningkatkan akurasi *FMST*.
3. Untuk meningkatkan akurasi hasil klastering, dapat dilakukan metode klastering berdasarkan *MST* yang lain.

[Halaman ini sengaja dikosongkan]

DAFTAR PUSTAKA

[1] The MathWorks, Inc, [Online]. Available: http://www.mathworks.com/discovery/pattern-recognition.html.

[2] Y. Sitohang. [Online]. Available: http://yukisuhendrasitohang.blogspot.com/2010/10/clustering-adalah-proses-mengelompokkan.html.

[3] P. Adhitama, "Kovloq [Never stop to (l)earn and share]," [Online]. Available: http://www.kovloq.com/2013/04/24/image-segmentation-using-k-means/.

[4] E. Barse, H. Kvarnström and E. Jonsson, *Synthesizing test data for fraud detection systems,* pp. 1-11, 2003.

[5] V. Estivill-Castro, "Why so many clustering algorithms," *Why so many clustering algorithms,* vol. 4, no. 1, pp. 65-75, 2002.

[6] Z. Caiming, M. Mikko, M. Dioqian and F. Pasi, "A Fast Minimum Spanning Tree Algorithm based on K-Means," 2014.

[7] P. K.Jana and A. Naik, "An Efficient Minimum Spanning Tree based Clustering Algorithm," 2009.

[8] D. Rachmawati, "Dee's Personal Pages," [Online]. Available: https://dee83.wordpress.com/2008/08/16/konsep-dasar-graph/.

[9] Saluky, "Saluky @etunas Best Solution," [Online]. Available: http://saluky.blogspot.com/2012/04/teori-graf.html.

[10] D. Rahardi, "SCIENCE FOR HUMANITY : Minimal Spanning Tree," [Online]. Available: http://dickyrahardi.blogspot.com/2008/05/minimal-spanning-tree.html.

[11] M. Fellows and N. Casey, "MegaMath," [Online]. Available: http://www.c3.lanl.gov/mega-math/gloss/graph/grpath.html.

[12] I. Firmansyah, "Algoritma Prims," [Online]. Available: https://sikomku.wordpress.com/2011/02/23/algoritma-prims/.

[13] Wolfram Research, Inc. , "Convex Set --from Mathworld Wolfram," [Online]. Available: http://mathworld.wolfram.com/ConvexSet.html.

[14] Encyclopædia Britannica, Inc, [Online]. Available: http://www.britannica.com/EBchecked/topic/135838/convex-set/images-videos/1737/convex-set-euclidean-geometry.

[15] GeeksforGeeks, "Greedy Algorithms | Set 5 (Prim's Minimum Spanning Tree (MST))," [Online]. Available: http://www.geeksforgeeks.org/greedy-algorithms-set-5-prims-minimum-spanning-tree-mst-2/.

[16] F. Dita, "Tahap-tahap K-Means Clustering," [Online]. Available: https://fadlikadn.wordpress.com/2013/06/14/tahap-tahap-k-means-clustering/.

[17] S. Towers, "Polymatheia," [Online]. Available: http://sherrytowers.com/2013/10/24/k-means-clustering/.

[18] E. Irwansyah, in *Advanced Clustering: Teori dan Aplikasi*, Sleman, DEEPUBLISH, 2015, p. 17.

[19] "Análisis y Manejo de Información," [Online]. Available: http://analisisymanejodeinformacion.blogspot.com/2011_12_08_archive.html.

[20] Ismail, "Berbagi Cerita & Kebahagiaan," [Online]. Available: https://basayevsmile.files.wordpress.com/2008/09/bab-6-visualisasi-data.pdf.

[21] National Science Foundation, [Online]. Available: http://archive.ics.uci.edu/ml/datasets.html.

[22] Wikipedia, "Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Iris_flower_data_set.

[Halaman ini sengaja dikosongkan]

BAB VI
KESIMPULAN DAN SARAN

Bab ini membahas mengenai kesimpulan yang dapat diambil dari hasil uji coba yang telah dilakukan sebagai jawaban dari rumusan masalah yang dikemukakan. Selain kesimpulan, juga terdapat saran yang ditujukan untuk pengembangan penelitian lebih lanjut.

6.1 Kesimpulan

Dari hasil uji coba yang telah dilakukan terhadap pembuatan model, dapat diambil kesimpulan sebagai berikut:

1. Metode *Fast Minimum Spanning Tree* (*FMST*) dapat digunakan untuk menyelesaikan *MST* lebih cepat dibanding algoritma *MST* konvensional dengan *weight error* rata-rata 0.988%.
2. Klastering berdasarkan *FMST* mendapatkan hasil terbaik ketika menggunakan *threshold*= 0.1 dan *step_size*= 0.1.
3. Klastering menggunakan *FMST* mampu mengelompokkan dataset yang berbentuk n*on-convex.* Akan lebih optimal jika data memiliki jarak antar klaster dan kerapatan data cukup tinggi. Akan tetapi, *FMST* kurang optimal digunakan mengelompokkan data yang memiliki *overlap* tinggi.

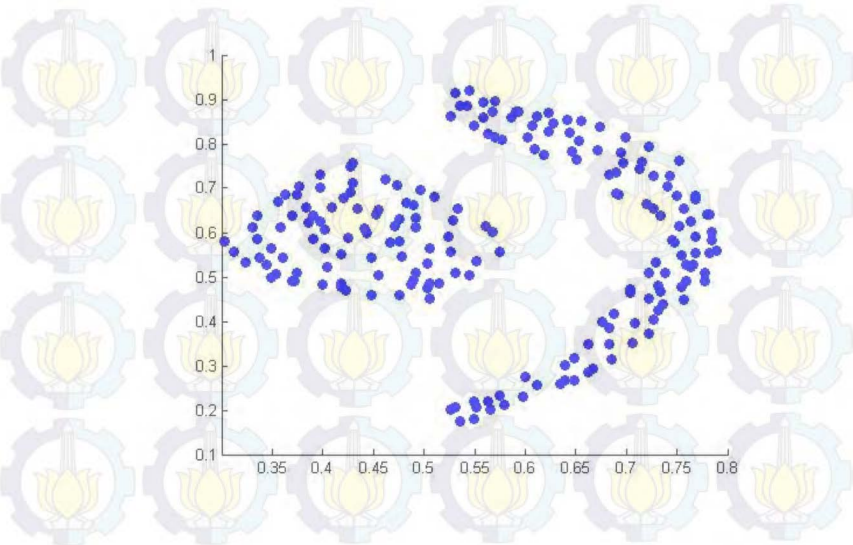
6.2 Saran

Saran yang diberikan untuk pengembangan aplikasi ini adalah:

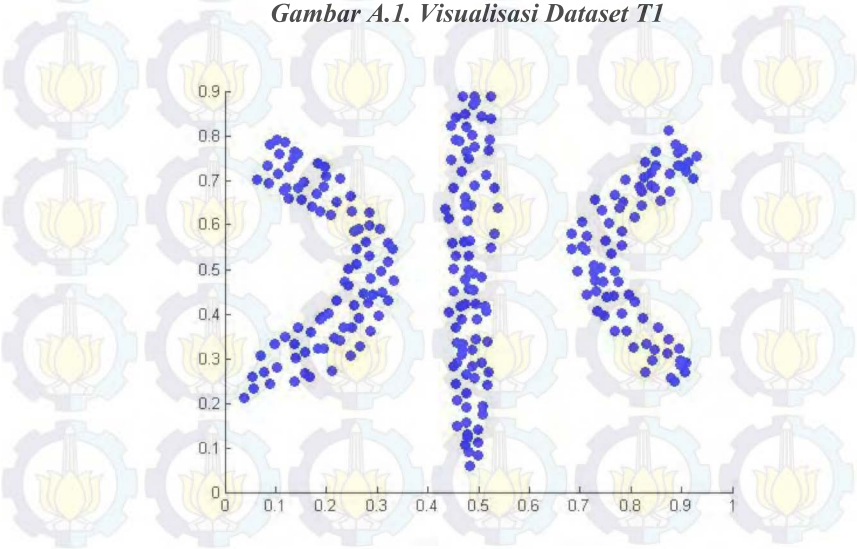
1. Untuk meningkatkan waktu pemrosesan, dapat digunakan bahasa pemrograman yang lain.
2. Perlu penelitian lebih lanjut mengenai tahap *Combine Subset Algorithm* untuk meningkatkan akurasi *FMST*.
3. Untuk meningkatkan akurasi hasil klastering, dapat dilakukan metode klastering berdasarkan *MST* yang lain.

[Halaman ini sengaja dikosongkan]

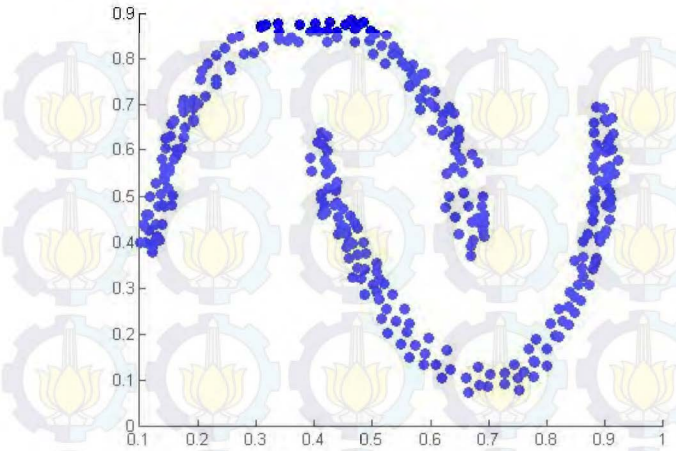
LAMPIRAN



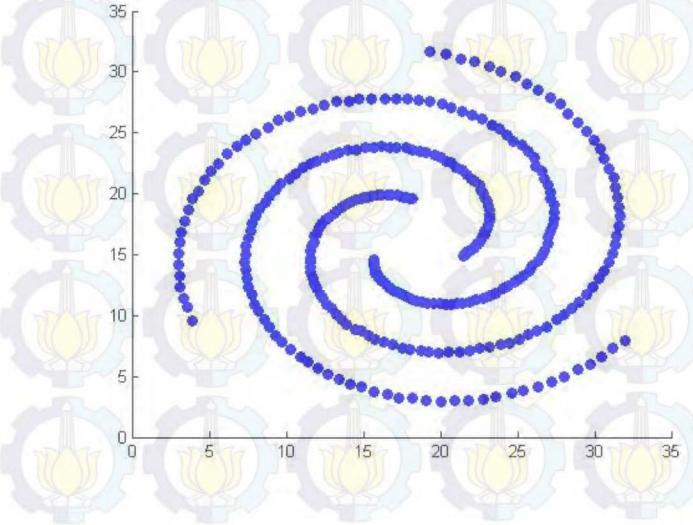
Gambar A.1. Visualisasi Dataset T1



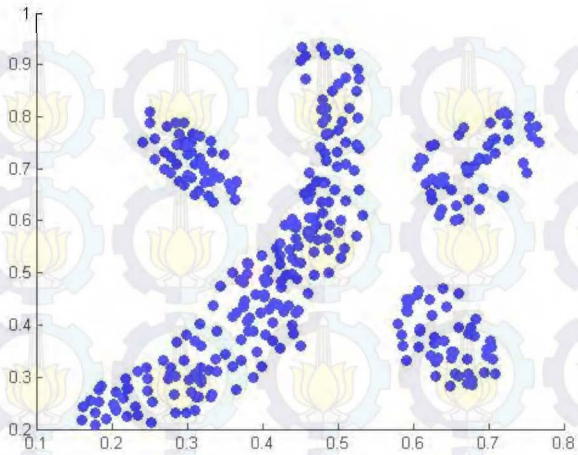
Gambar A.2. Visualisasi Dataset T2



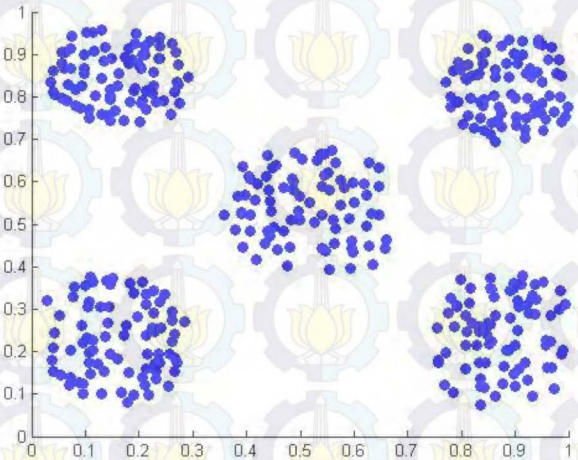
Gambar A.3. Visualisasi Dataset Curve



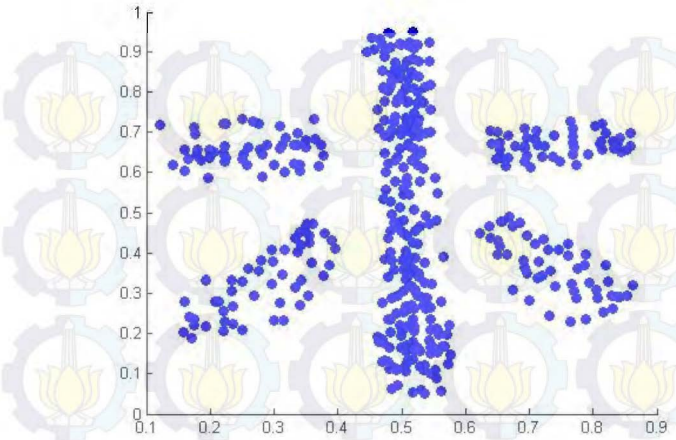
Gambar A.4. Visualisasi Dataset Spiral1



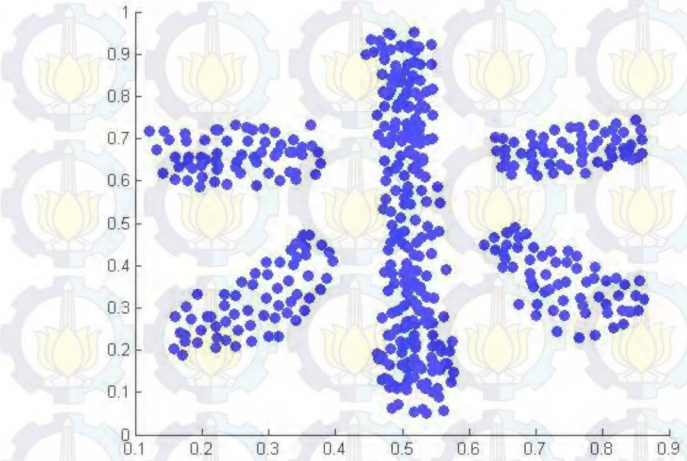
Gambar A.5. Visualisasi Dataset T3



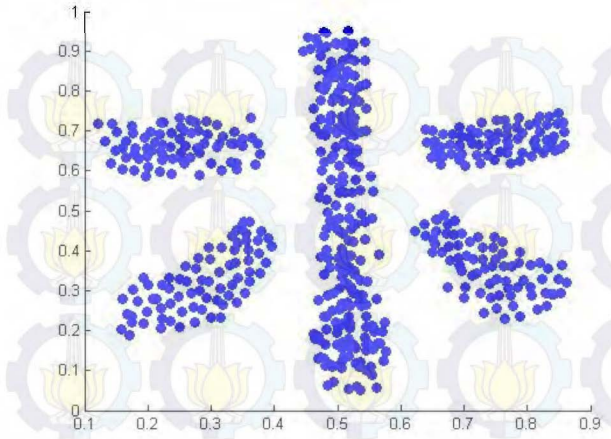
Gambar A.6. Visualisasi Dataset T4



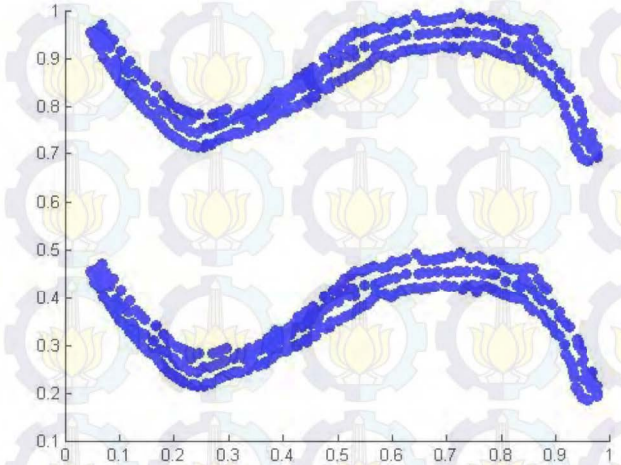
Gambar A.7. Visualisasi Dataset T5



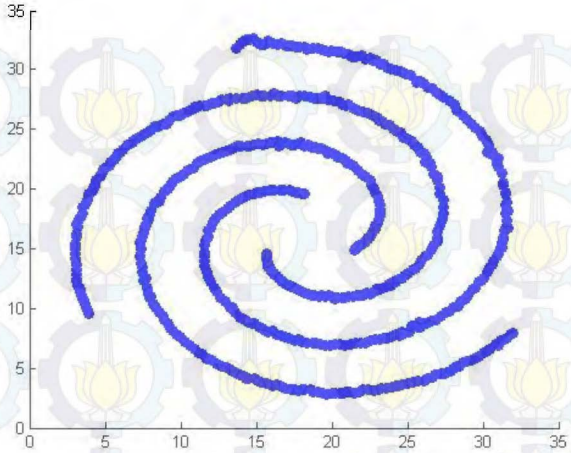
Gambar A.8. Visualisasi Dataset T6



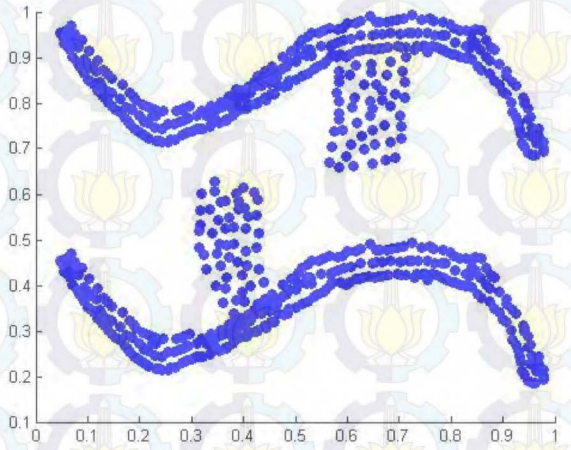
Gambar A.9. Visualisasi Dataset T7



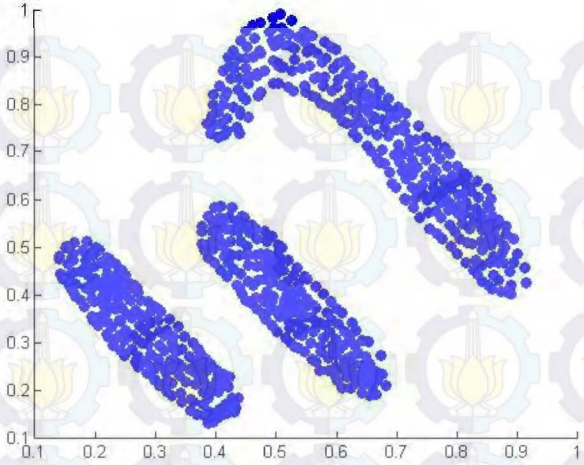
Gambar A.10. Visualisasi Dataset T8



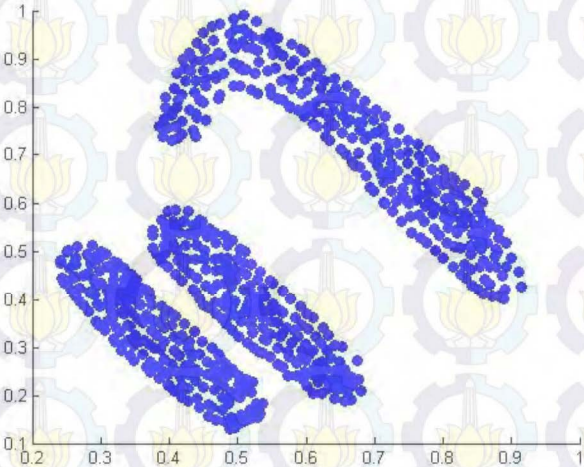
Gambar A.11. Visualisasi Dataset Spiral2



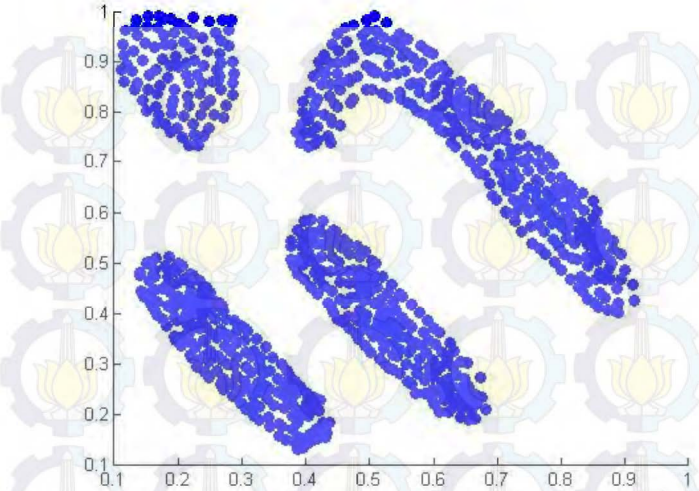
Gambar A.12. Visualisasi Dataset T9



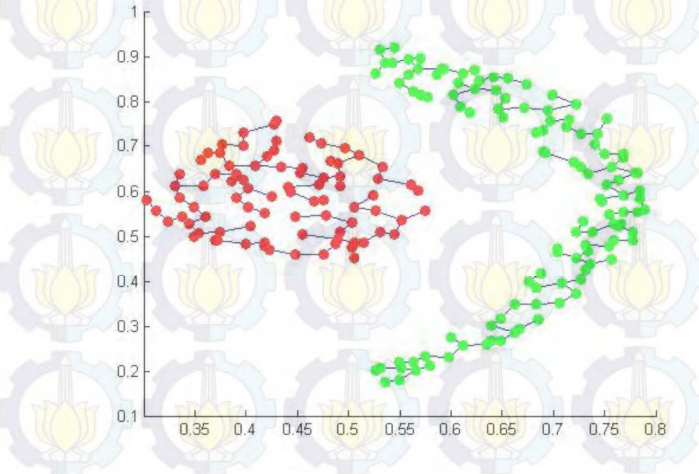
Gambar A.13. Visualisasi Dataset T10



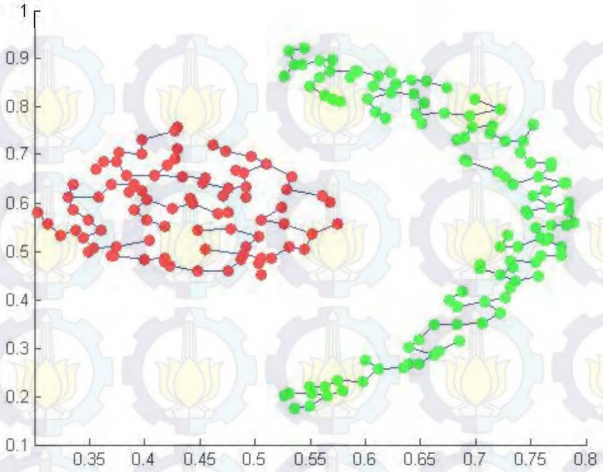
Gambar A.14. Visualisasi Dataset T11



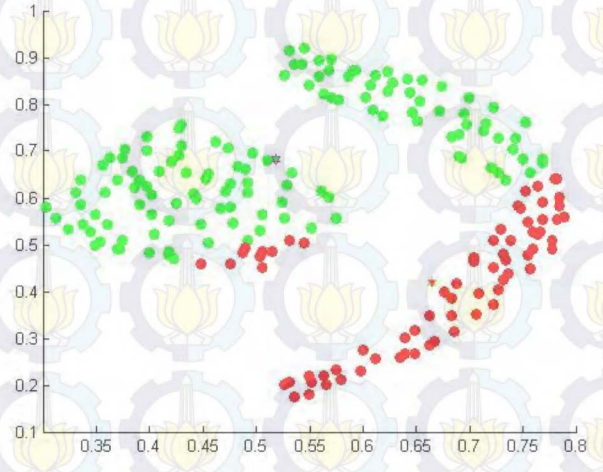
Gambar A.15. Visualisasi Dataset T12



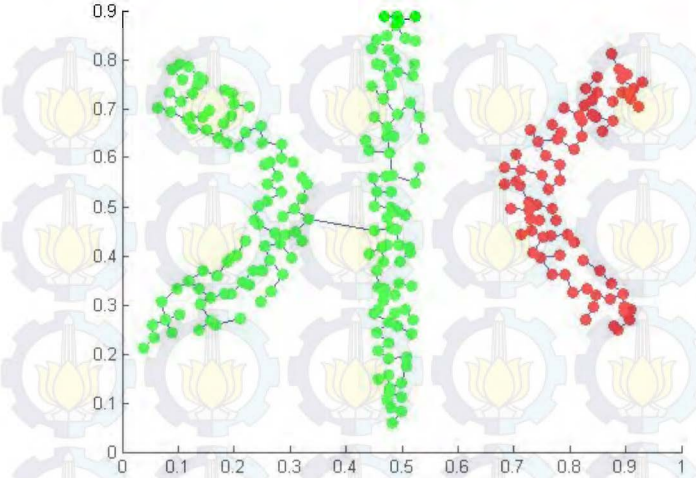
Gambar A.16. Hasil Uji Coba 1 menggunakan FMST pada percobaan ke-1



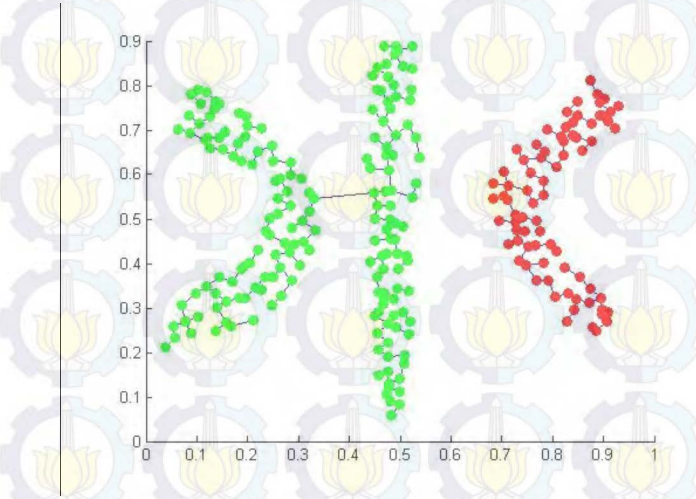
Gambar A.17. Hasil Uji Coba 1 menggunakan MST pada percobaan ke-1



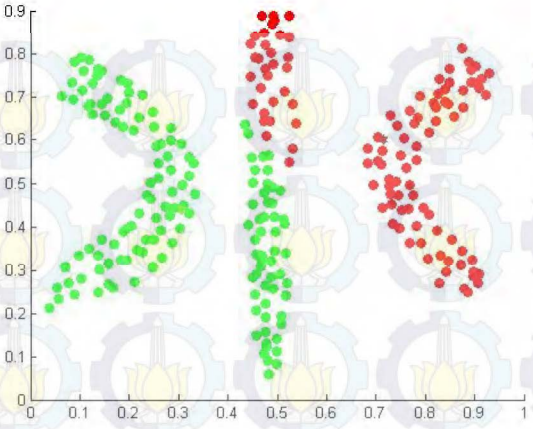
Gambar A.18. Hasil Uji Coba 1 menggunakan Kmeans pada percobaan ke-1



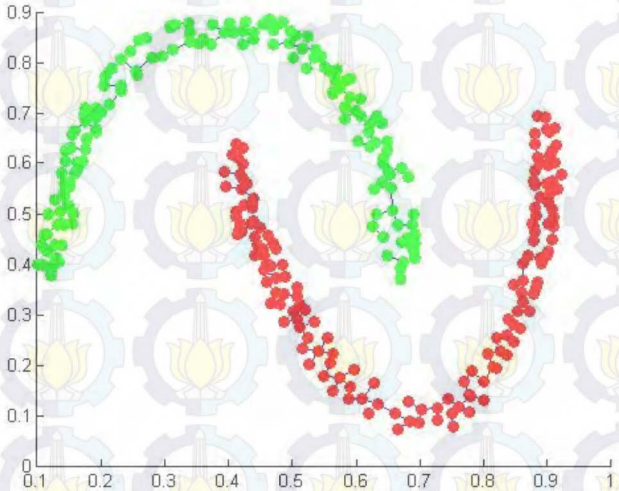
Gambar A.19. Hasil Uji Coba 1 menggunakan FMST pada percobaan ke-2



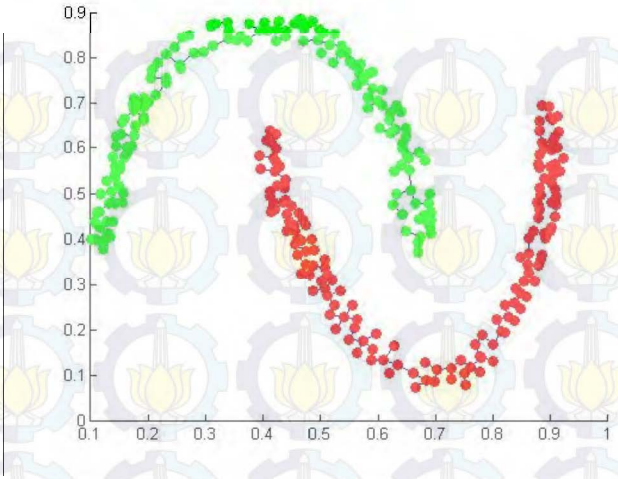
Gambar A.20. Hasil Uji Coba 1 menggunakan MST pada percobaan ke-2



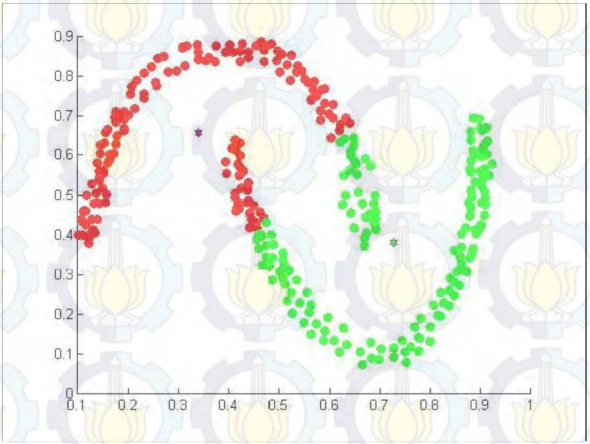
Gambar A.21. Hasil Uji Coba 1 menggunakan Kmeans pada percobaan ke-2



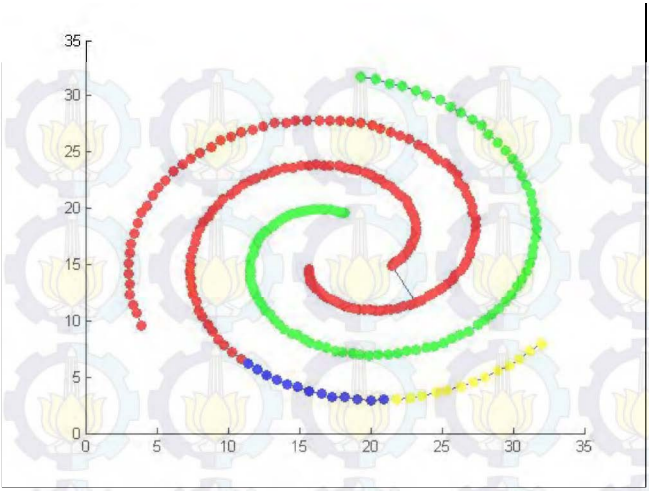
Gambar A.22. Hasil Uji Coba 1 menggunakan FMST pada percobaan ke-3



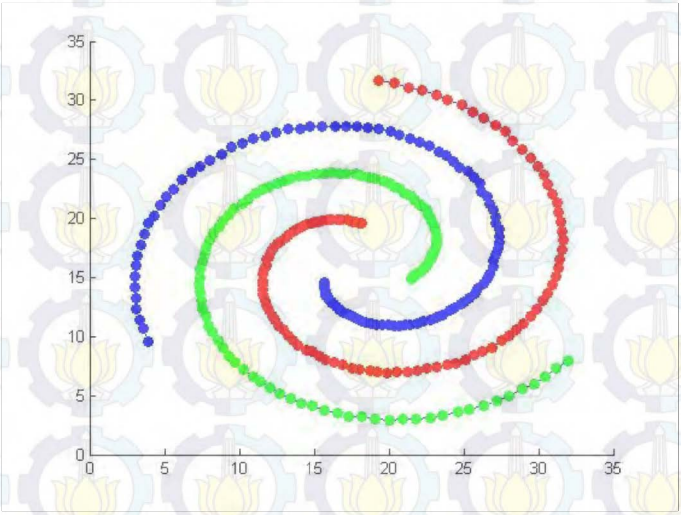
Gambar A.23. Hasil Uji Coba 1 menggunakan MST pada percobaan ke-3



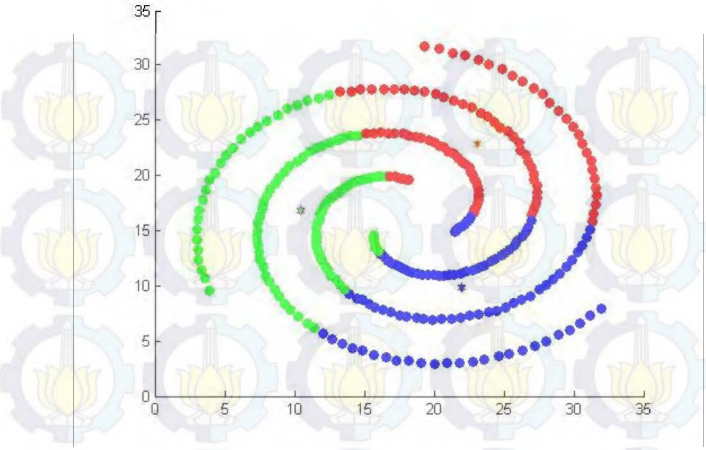
Gambar A.24. Hasil Uji Coba 1 menggunakan Kmeans pada percobaan ke-3



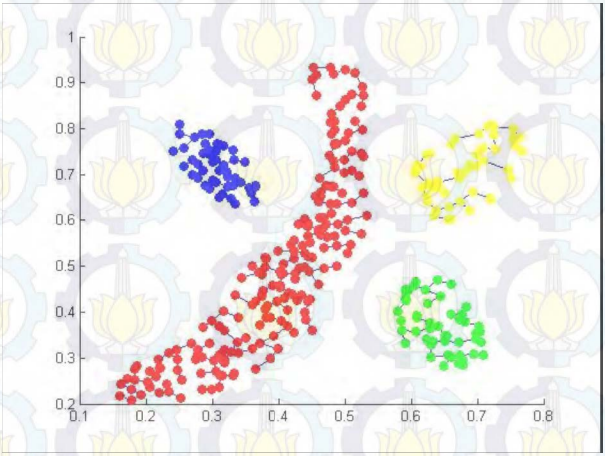
Gambar A.25. Hasil Uji Coba 1 menggunakan FMST pada percobaan ke-4



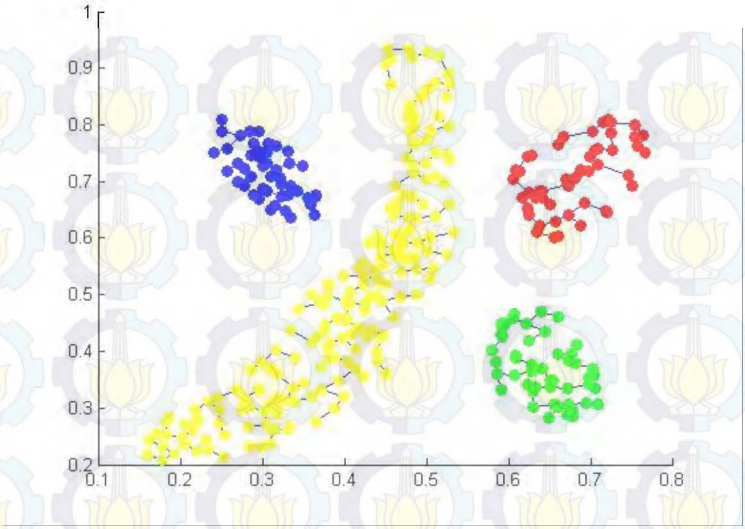
Gambar A.26 Hasil Uji Coba 1 menggunakan MST pada percobaan ke-4



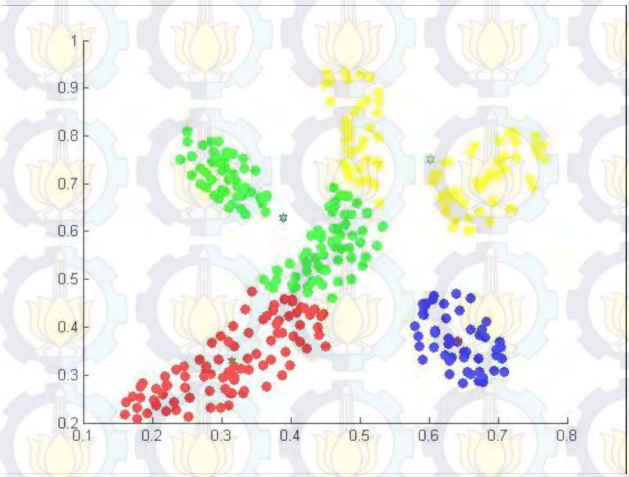
Gambar A.27. Hasil Uji Coba 1 menggunakan Kmeans pada percobaan ke-4



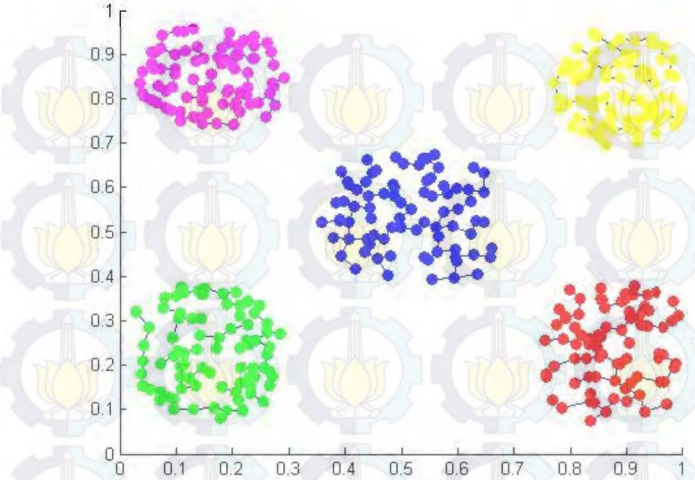
Gambar A.28. Hasil Uji Coba 1 menggunakan FMST pada percobaan ke-5



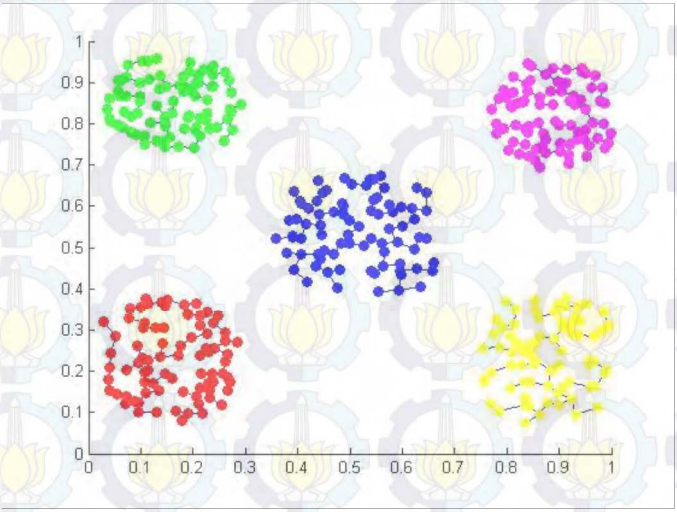
Gambar A.29. Hasil Uji Coba 1 menggunakan MST pada percobaan ke-5



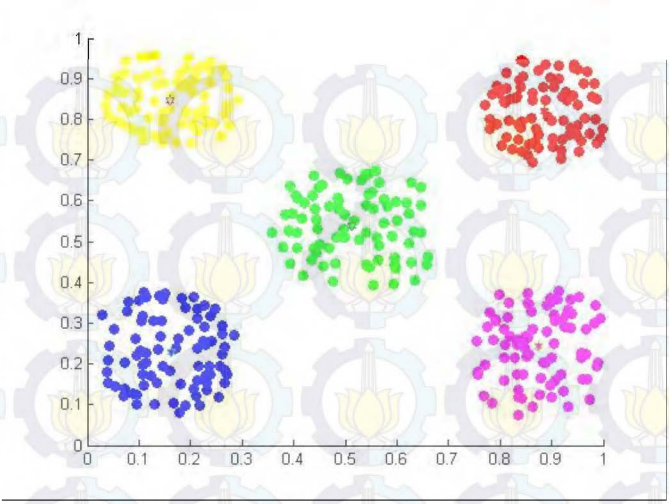
Gambar A.30. Hasil Uji Coba 1 menggunakan Kmeans pada percobaan ke-5



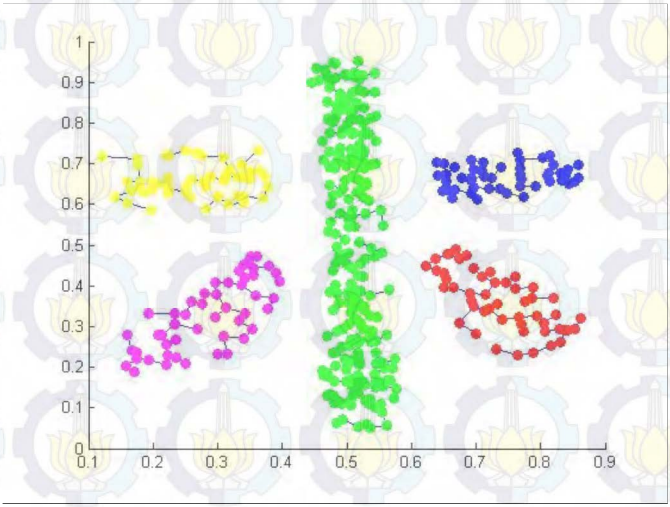
Gambar A.31. Hasil Uji Coba 1 menggunakan FMST pada percobaan ke-6



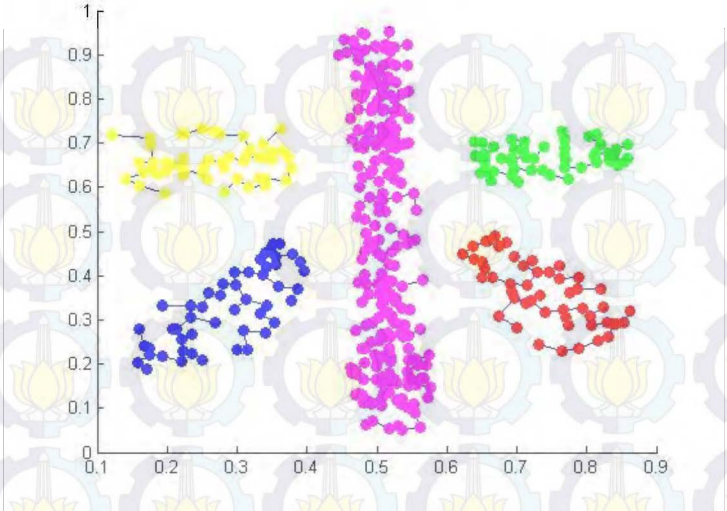
Gambar A.32. Hasil Uji Coba 1 menggunakan MST pada percobaan ke-6



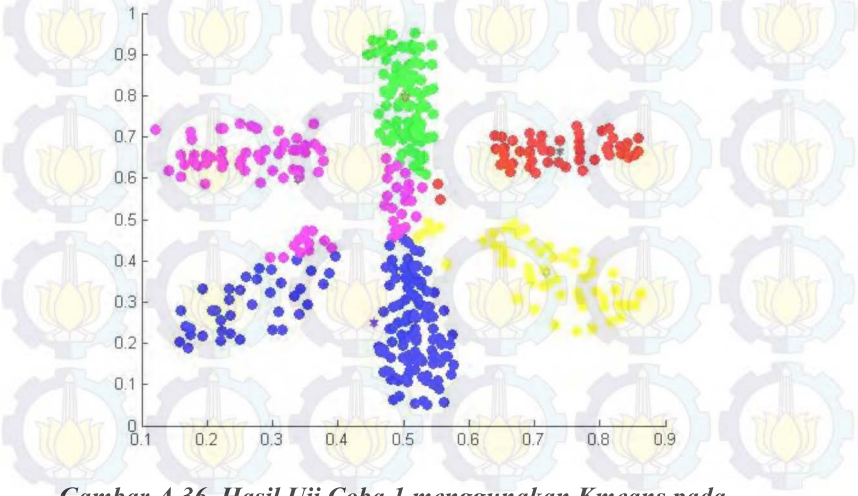
Gambar A.33. Hasil Uji Coba 1 menggunakan Kmeans pada percobaan ke-6



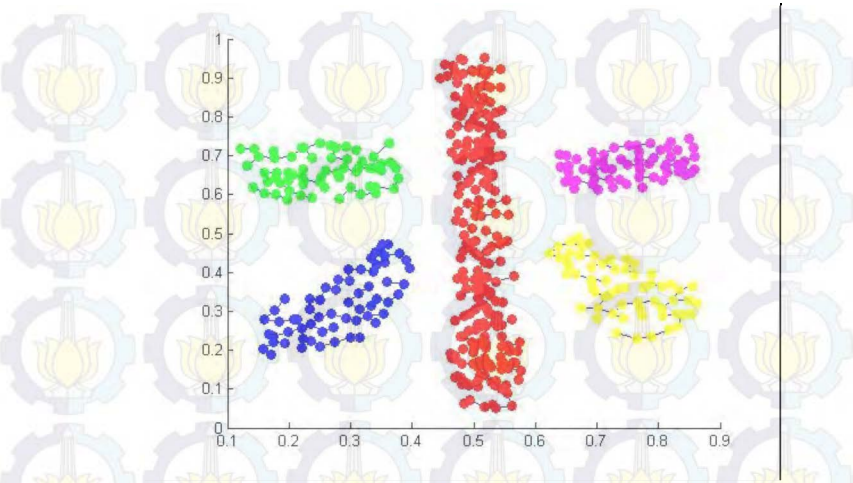
Gambar A.34. Hasil Uji Coba 1 menggunakan FMST pada percobaan ke-7



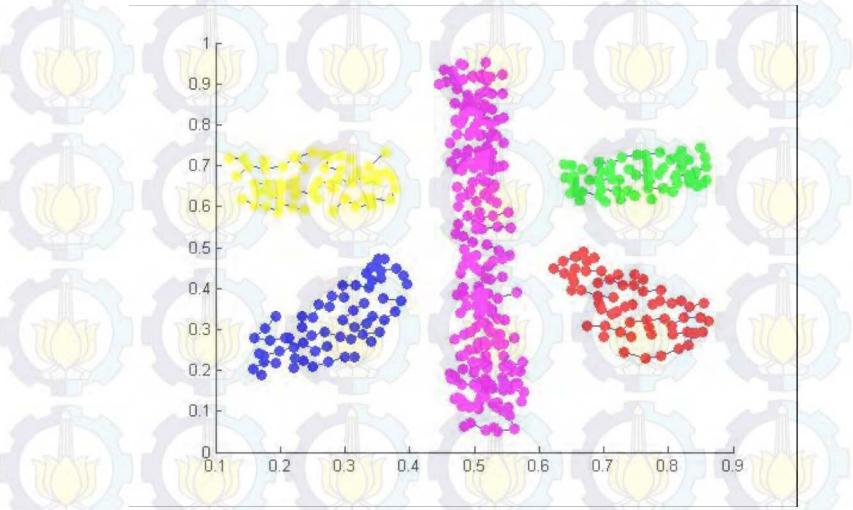
Gambar A.35. Hasil Uji Coba 1 menggunakan MST pada percobaan ke-7



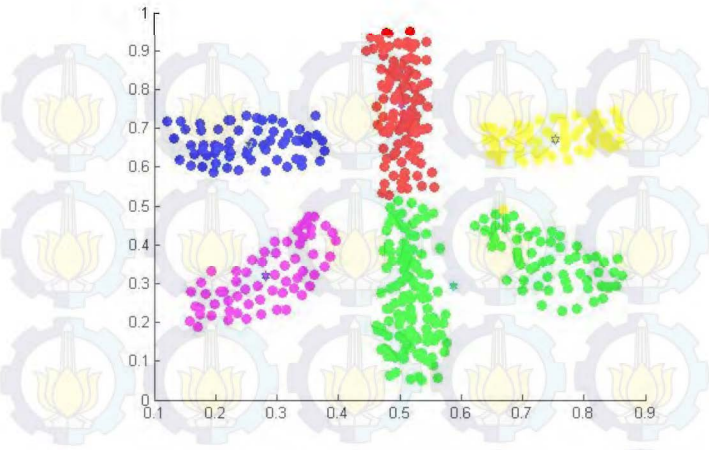
Gambar A.36. Hasil Uji Coba 1 menggunakan Kmeans pada percobaan ke-7



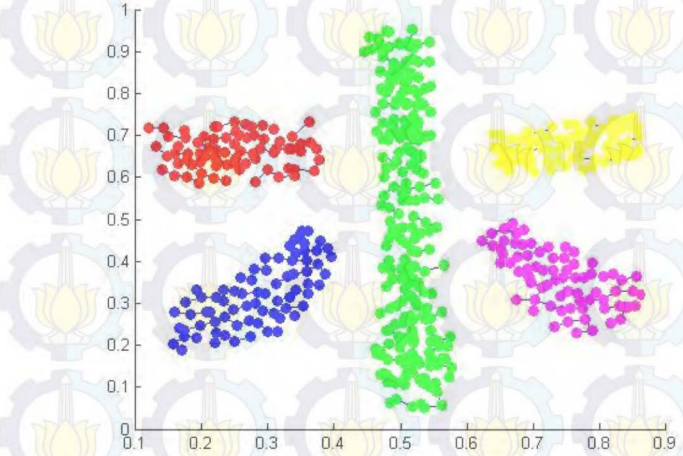
Gambar A.37. Hasil Uji Coba 1 menggunakan FMST pada percobaan ke-8



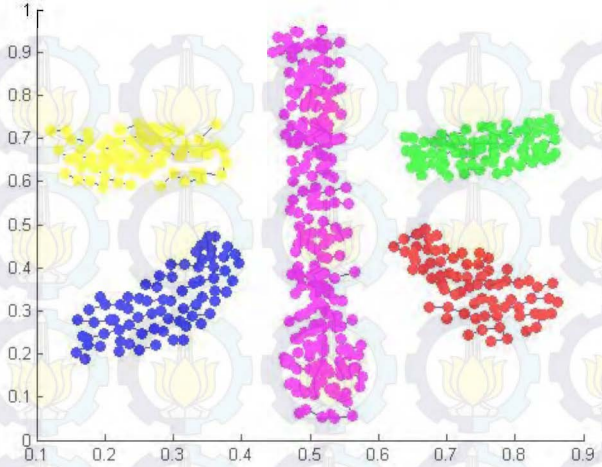
Gambar A.38. Hasil Uji Coba 1 menggunakan MST pada percobaan ke-8



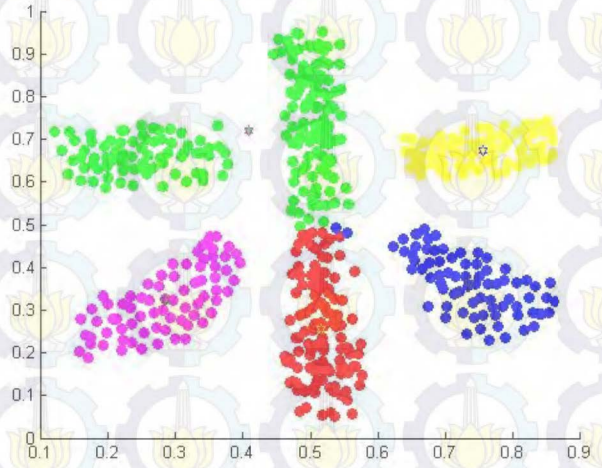
Gambar A.39. Hasil Uji Coba 1 menggunakan Kmeans pada percobaan ke-8



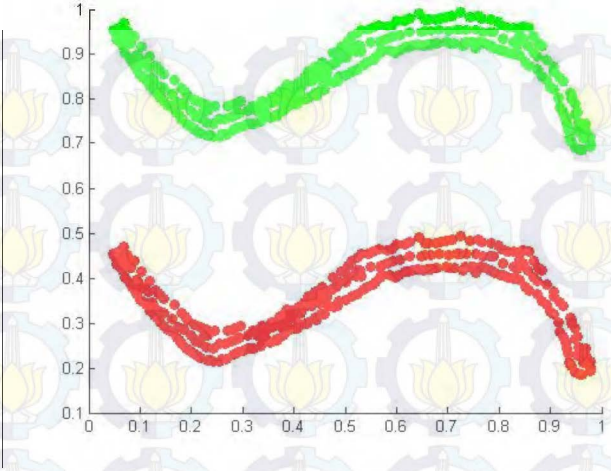
Gambar A.40. Hasil Uji Coba 1 menggunakan FMST pada percobaan ke-9



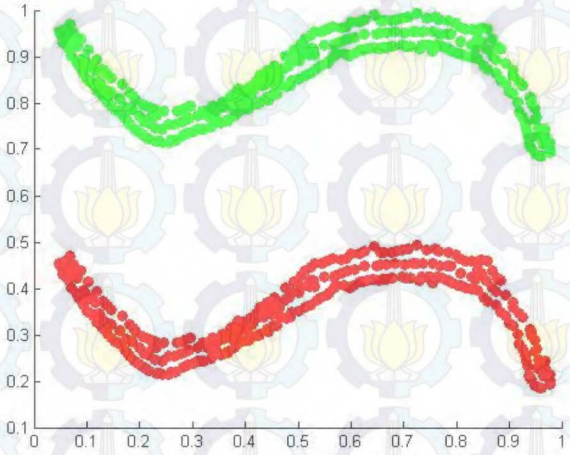
Gambar A.41. Hasil Uji Coba 1 menggunakan MST pada percobaan ke-9



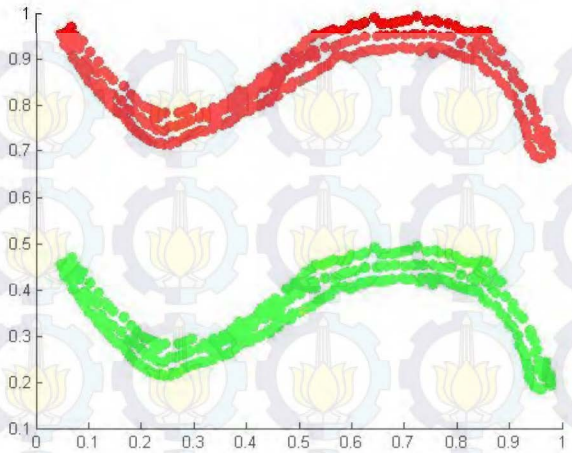
Gambar A.42. Hasil Uji Coba 1 menggunakan Kmeans pada percobaan ke-9



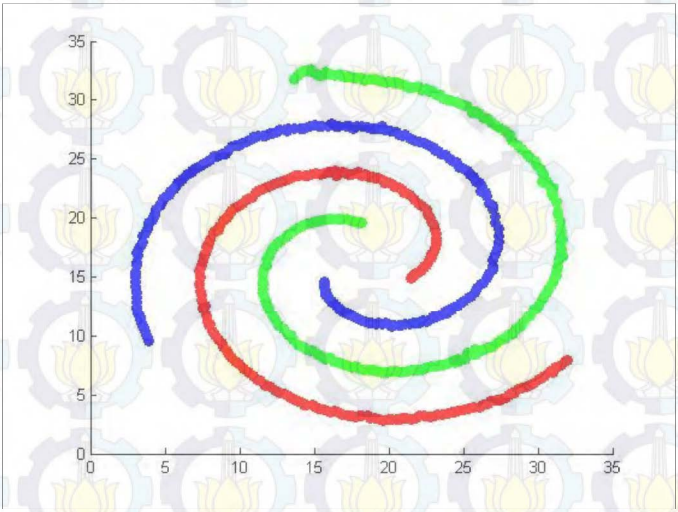
Gambar A.43. Hasil Uji Coba 1 menggunakan FMST pada percobaan ke-10



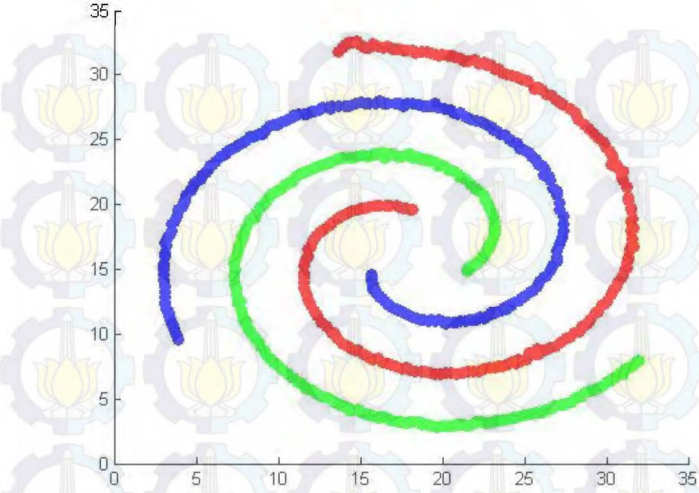
Gambar A.44. Hasil Uji Coba 1 menggunakan MST pada percobaan ke-10



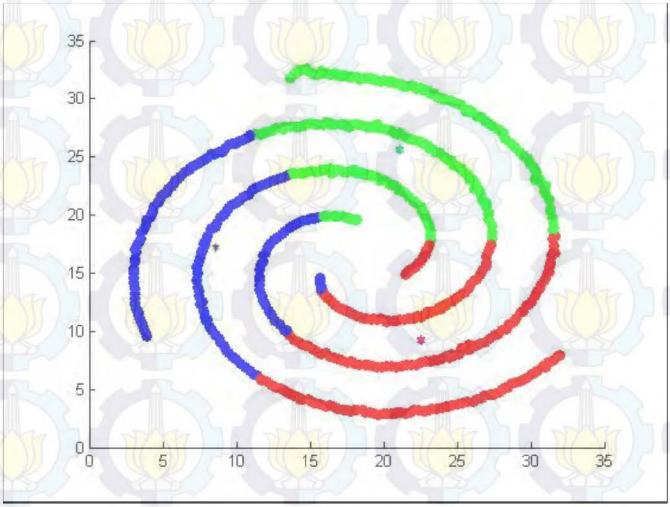
Gambar A.45. Hasil Uji Coba 1 menggunakan Kmeans pada percobaan ke-10



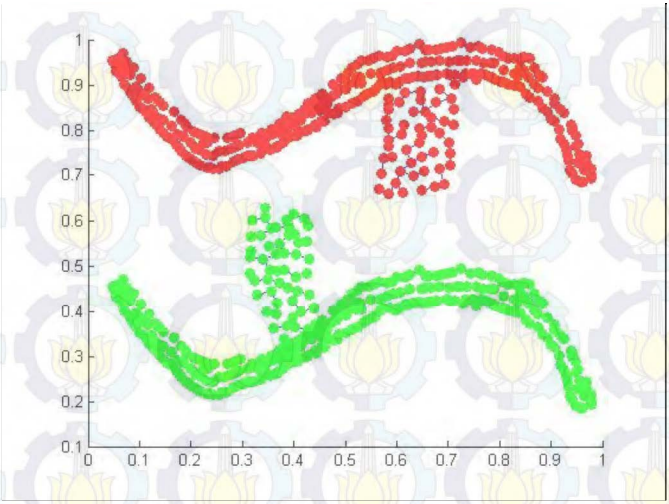
Gambar A.46. Hasil Uji Coba 1 menggunakan FMST pada percobaan ke-11



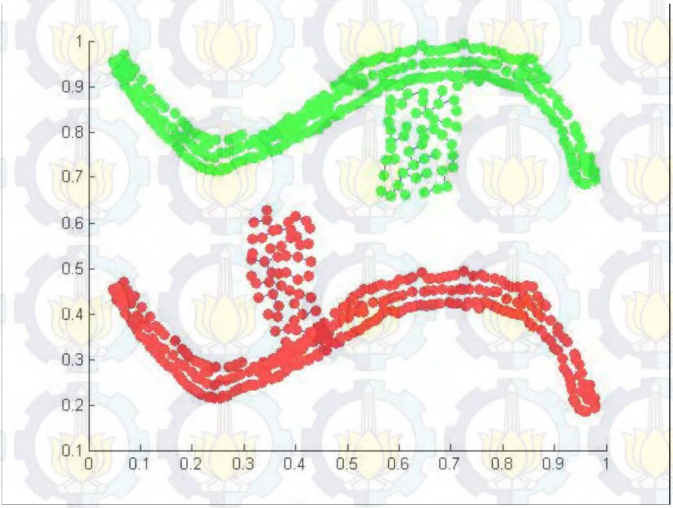
Gambar A.47. Hasil Uji Coba 1 menggunakan MST pada percobaan ke-11



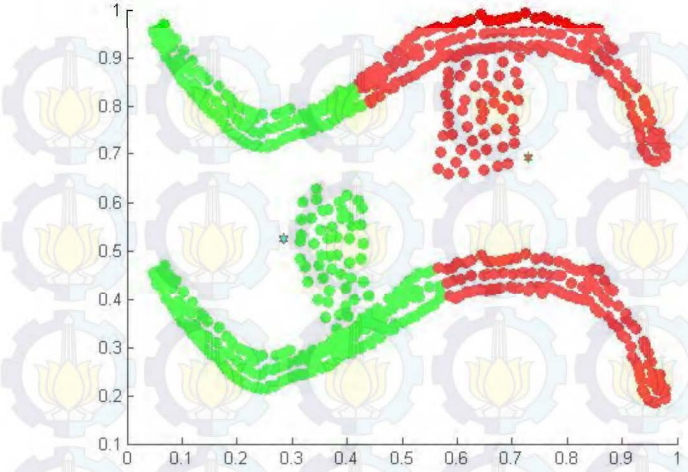
Gambar A.48. Hasil Uji Coba 1 menggunakan Kmeans pada percobaan ke-11



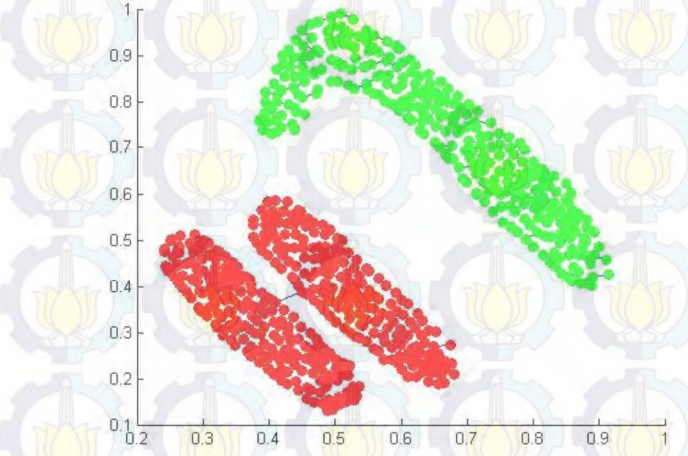
Gambar A.49. Hasil Uji Coba 1 menggunakan FMST pada percobaan ke-12



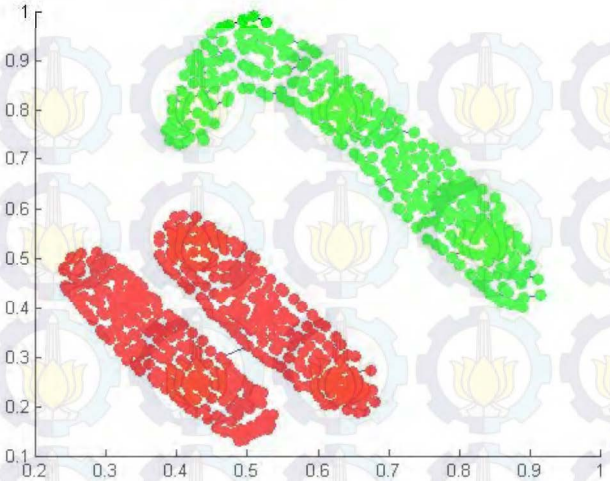
Gambar A.50. Hasil Uji Coba 1 menggunakan MST pada percobaan ke-12



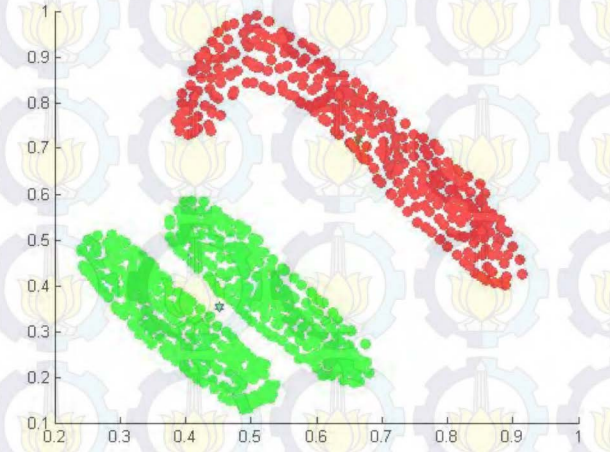
Gambar A.51. Hasil Uji Coba 1 menggunakan Kmeans pada percobaan ke-12



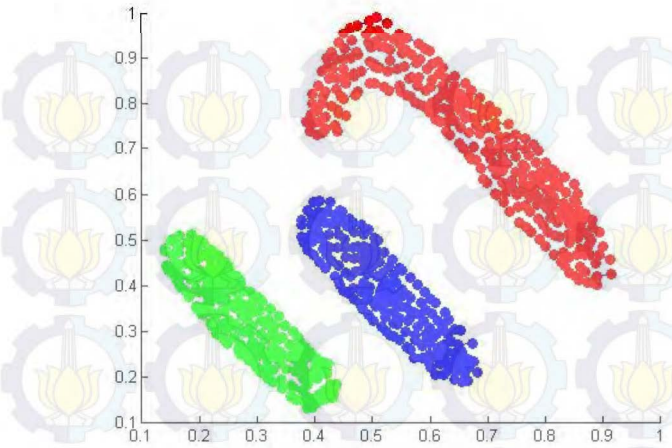
Gambar A.52. Hasil Uji Coba 1 menggunakan FMST pada percobaan ke-13



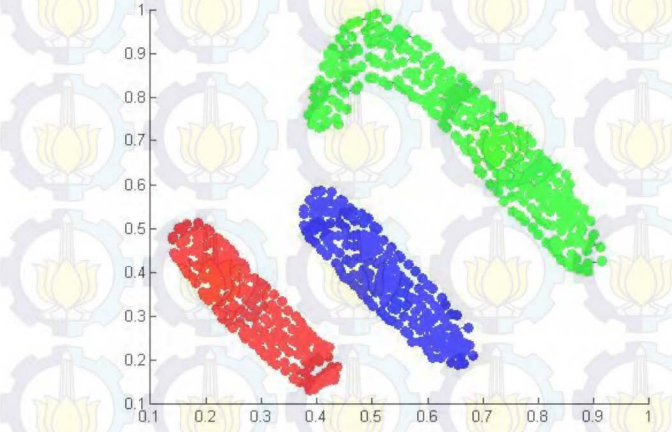
Gambar A.53. Hasil Uji Coba 1 menggunakan MST pada percobaan ke-13



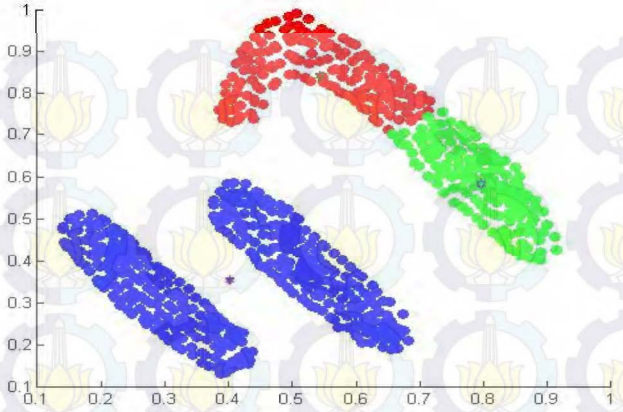
Gambar A.54. Hasil Uji Coba 1 menggunakan Kmeans pada percobaan ke-13



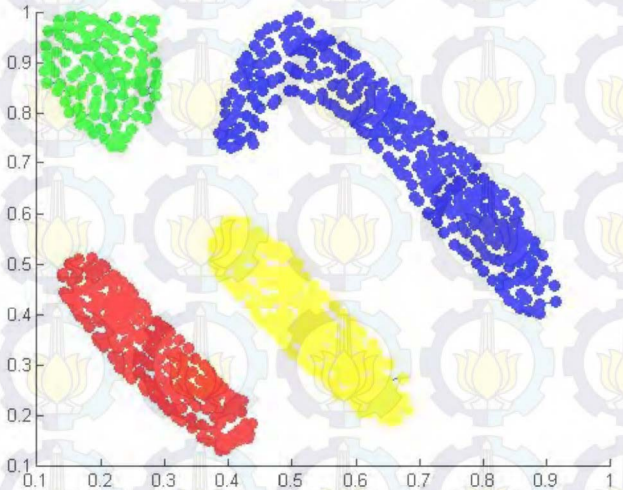
Gambar A.55. Hasil Uji Coba 1 menggunakan FMST pada percobaan ke-14



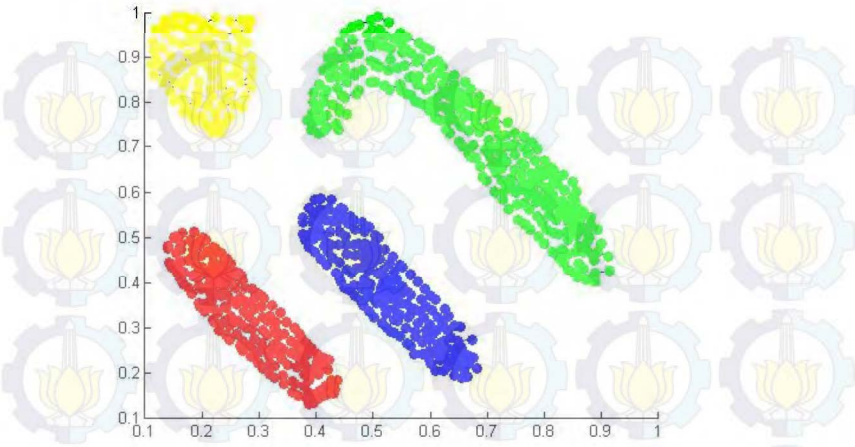
Gambar A.56. Hasil Uji Coba 1 menggunakan FMST pada percobaan ke-14



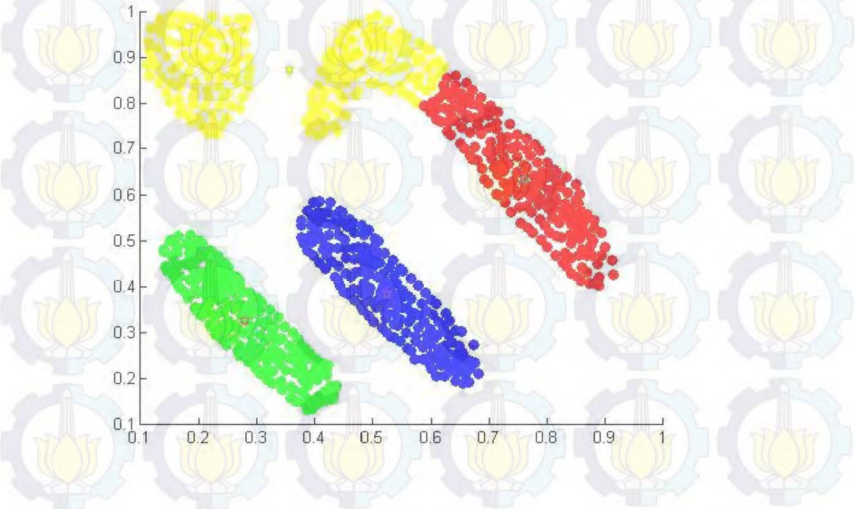
Gambar A.57. Hasil Uji Coba 1 menggunakan Kmeans pada percobaan ke-14



Gambar A.58. Hasil Uji Coba 1 menggunakan FMST pada percobaan ke-15



Gambar A.59. Hasil Uji Coba 1 menggunakan MST pada percobaan ke-15



Gambar A.60. Hasil Uji Coba 1 menggunakan Kmeans pada percobaan ke-15